Fine-tune a Neural Machine Translation Model for Legal Domain: theories explained and practice with Hugging Face

Author:

Guocheng Zhu

Supervisor:

Adrià Martín-Mor

Master's Degree Thesis

Official Master's Degree Tradumatics: Translation Technologies (2020 – 2021)

Universitat Autònoma de Barcelona

July 2021

Abstract: As the state-of-art of machine translation, neural machine translation is already established in many translation projects. It would be beneficial for translation service professionals to understand its mechanism to leverage it better. In this master thesis, we try to explain neural networks and neural machine translation in detail for the audience of non-scientific backgrounds, with a particular focus on the Transformer model. Also, we present a model training practice with Python to showcase how different concepts in NMT are applied to the training process. The training leverages a pre-trained NMT model to adapt its domain from general to legal, for which we also constructed an in-domain parallel corpus manually with resources from various legal databases and other available texts. The fine-tuned model achieves an increment of 6.9 on the BLEU score and outperforms the original model in various aspects.

Keywords: NMT, Transformer, domain adaptation, Spanish to Chinese

Resum: Com el mètode més avançat de la traducció automàtica, la traducció automàtica neuronal ja està establerta en molts projectes de traducció. Seria beneficiós per als professionals dels serveis de traducció comprendre el seu mecanisme per aprofitar-lo millor. En aquest treball de fi de màster, tractem d'explicar les xarxes neuronals i la traducció automàtica neuronal en detall per a l'audiència d'antecedents no científics, amb un enfocament particular en el model *Transformer*. A més, presentem una pràctica d'entrenament amb Python per mostrar com s'apliquen diferents conceptes de la TAN al procés d'entrenament. La pràctica aprofita un model TAN prèviament entrenat per adaptar el seu domini de general a legal, per això també vam construir manualment un corpus paral·lel en el domini amb recursos de diverses bases de dades legals i altres textos disponibles. El model optimitzat aconsegueix un increment de 6,9 en la puntuació BLEU i supera el model original en diversos aspectes.

Paraules clau: TAN, *Transformer*, adaptació de domini, castellà a xinès

Resumen: Como el método más avanzado de la traducción automática, la traducción automática neuronal ya está establecida en muchos proyectos de traducción. Sería beneficioso para los profesionales de los servicios de traducción comprender su mecanismo para aprovecharlo mejor. En este trabajo de fin de máster, tratamos de explicar las redes neuronales y la traducción automática neuronal en detalle para la audiencia de antecedentes no científicos, con un enfoque particular en el modelo *Transformer*. Además, presentamos una práctica de entrenamiento con Python para mostrar cómo se aplican diferentes conceptos de la TAN al proceso de entrenamiento. La práctica aprovecha un modelo TAN previamente entrenado para adaptar su dominio de general a legal, para lo cual también construimos manualmente un corpus paralelo en el dominio con recursos de varias bases de datos legales y otros textos disponibles. El modelo optimizado logra un incremento de 6,9 en la puntuación BLEU y supera al modelo original en varios aspectos.

Palabras clave: TAN, *Transformer*, adaptación de dominio, español a chino

Acknowledgement

I would like first to acknowledge the precious guidance my supervisor Adrià Martín-Mor gave me along all stages of writing this master thesis. Also, I shall express my gratitude to my teacher ZHANG Tianqi, who helped me academically in many ways. I would also like to thank HAN Jingyi for her recommendations on my research.

The completion of this master thesis could not have been possible without the great help of my friend WANG Rouyou. A debt of gratefulness is also owed to Jay Alammer, WU Jie, and ZHANG Yi for their valuable answers to my inquiries. Sincere thanks to Alfonso Sánchez Romera as well for reviewing the Catalan abstract.

In addition, I genuinely thank the support and love of all my dear friends, especially Sergio Suárez, without whom the journey would have been unbearable, even though they were not always aware of it.

Last but not least, many thanks to my parents for supporting me financially, particularly to my mother with her unconditional love and being my hero.

Contents

Table of Figures

1. Introduction

Translation is an important activity in human societies that has been conducted for centuries. It is not surprising that when computers, or even the idea of computers, came into existence, how to use the machine to translate quickly constituted a significant research area (Koehn, 2020, pp. 33–34). The current output quality of machine translation systems like Google Translate often amazes and terrifies human translators because they are afraid of being substituted by machines. Machine translation indeed has altered translators' specific functions in the language service industry, but it is not necessarily terrifying.

On the one hand, machine translation still needs human translators to edit the outputs, ensuring that the final translation quality is deliverable. On the other hand, the fusion of machine and human in the language sector also creates new professional opportunities, such as corpus cleaning, machine translation quality analysis, and machine translation engine training. That said, it is beneficial for translators or other language service sector professionals to understand how machine translation works in order to leverage it.

There are several approaches to machine translation: rule-based, example-based, statistical method, and neural networks. They fall into two major categories: rule-based and corpus-based (also called data-driven) (Forcada, 2017). Neural machine translation is the youngest but the most promising approach. It is also depicted as challenging to understand, particularly due to those concise mathematical equations in academic papers. Besides, these papers often assume that the audience is familiar with machine learning jargons and concepts, which sets a high barrier for common language industry professionals to understand NMT in detail.

This master thesis tries to walk the readers through frequent concepts related to neural networks and NMT and explain them thoroughly. Although the author intents to deconstruct abstract mathematical equations into small steps and graphs, it is still recommended that the readers have some basic knowledge of linear algebra and calculus. The author will also explain how he fine-tuned a NMT model using Hugging Face, a community, company, and Python libraries in natural language processing.

Hopefully, this thesis itself could constitute a more approachable explanation of neural machine translation and a hands-on guide of its training. Note that this thesis focuses on the Spanish and Simplified Chinese language pair due to the author's linguistic background and personal interest.

First, in Chapter 2, we will delve into the fundamentals of neural networks and principles of neural machine translation. Then, in Chapter 3, we will present the methodology of our experiment to fine-tune a neural machine translation model from the general domain to the legal domain, including the introduction to some major toolkits used and how we constructed a custom parallel corpus. We will also comment briefly on the training process and compare between the original and the fine-tuned model in this chapter. Finally, Chapter 4 summarizes major results and reflections on the experiment as well as possible future work to be done.

2. Theoretical Framework

This chapter provides various principles behind neural machine translation with particular attention to the Transformer model (See Section 2.4). It helps the reader familiarize with frequently occurring terms and concepts of neural networks, have an overview of adjustable hyperparameters, and understand how these adjustments could affect the neural machine translation model. This chapter gives the audience the fundamentals to train neural machine translation systems, especially with toolkits of a relatively high level of customization.

After a brief introduction to major machine translation approaches, we will first explore the main mechanisms inside the black box of neural networks. Then the discussion will move to neural networks adaptation to machine translation, covering word embeddings, recursive neural network architecture, and attention mechanism. Later, we will explain how Transformer centers on self-attention instead of recursiveness to achieve a new state-of-the-art. Finally, in Section 2.5, there will be a reflection on machine translation evaluation concerning its possible usage issues, as a reminder of appropriate attitude towards neural machine translation and its improvable margins.

2.1. Machine Translation Approaches

Machine translation (**MT**) does not mean a total separation of human interaction, but its essence is the automation of interlingual conversion of text (Baker & Saldanha, 2019, pp. 305–306; Koehn, 2020, p. 36).

Dating from the WWII, machine translation has gone through various revolutions. In the beginning, machine translation is rule-based. Humans coded linguistic rules into computers to conduct morphological and syntactic analysis on source text and then reconstruct it in the target language, using monolingual and bilingual dictionaries and computational grammar modules (known as the Transfer). We can see that **rule-based machine translation** (**RBMT**) resembles how humans translate, but it requires continuously coding of linguistic rules and maintaining the system (Sánchez Ramos & Rico Pérez, 2020, p. 15).

It is worth mentioning the Vauquois triangle here. It summarizes and classifies machine translation approaches at the time of the article (Bernard Vauquois, 2003). In the article, Vauquois traverses two generations of machine translation and proposes this linguistical architecture of different levels (Chéragui, 2012). He views different machine translation approaches within the same framework of three steps: analyze, transfer, and generation. Different subcategories of RBMT, such as direct translation, morphological transfer, syntactic transfer, and semantic transfer, all fall into this framework (Sánchez Ramos & Rico Pérez, 2020). As the level of understanding of the language for machine rises, it will finally reach a point called interlingua where the transfer process is no longer needed and ideally be suitable for multilingual transduction among languages (Bernard Vauquois, 2003; Chéragui, 2012).

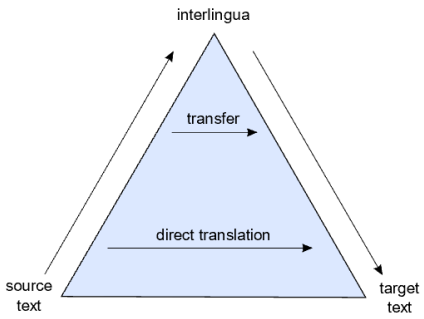

Figure 2.1 Illustration of Vauquois triangle[1].

[1] Accessed from https://en.wikipedia.org/wiki/Transfer-based_machine_translation#/media/File:Direct_translation_and_transfer_translation_pyramid.svg

Starting from the 80s, researchers drifted away from this framework and began to use bilingual corpora and probability theory to approach MT (Koehn, 2020, p. 36). This data-driven method, also called **statistical machine translation** (**SMT**), facilitated MT system development and often exceeded RBMT performance as it exploited human translated text (Baker & Saldanha, 2019, pp. 306–307). SMT is essentially distinct from RBMT because it abandoned the idea of teaching computers any linguistic information (Sánchez Ramos & Rico Pérez, 2020, p. 16); instead, it relied on translators' work. The translation was not generated linguistically but more like a collage made from translated bilingual segments.

Neural machine translation (**NMT**) is also a data-driven method. It can be viewed together with SMT as a corpus-based approach for machine translation (Chéragui, 2012). NMT resembles SMT in that it also exploits bilingual corpora and resorts probability but differs in that NMT uses neural networks to train the system.

There are relatively fewer academic papers aiming to explain NMT in depth to students or professionals from translation studies field. "Making Sense of Neural Machine Translation" is an attempt, in which Forcada introduces some basic concepts of this MT approach and how it works (Forcada, 2017). This paper is introductory in that it does successfully avoid mathematical complexity but also loses many details of the training process, such as "vocabulary size", "optimizer", "learning rate", etc. They are actually important concepts if you try to train a productive NMT system in real life. A preprint chapter (Pérez-Ortiz et al., 2021) of the upcoming textbook "Machine Translation for Multilingual Citizens" from the MultiTraiNMT project[2] goes further in this direction and adds more technical close-ups to neural machine translation. Besides, there are not many researches on the Chinese < > Spanish language pair for neural machine translation either. One noteworthy paper is Costa-Jussà, Aldón, and Fonollosa´s experiment of incorporating Chinese characters or words bitmap font information into word embeddings (see Section 2.3.1) (Costa-Jussà et al., 2017).

2.2. Neural Networks

Although it seems that it is not until recent years when neural networks gain popularity in the general public, their conception dates back to the 1940s. Since then, it has never

[2] https://multitrainmt.eu

disappeared from academia (Yadav et al., 2015). This section gives a condensed presentation of the essentials of neural network design and training.

2.2.1. Relationship between neural networks and other concepts

Artificial neural networks, simply called **neural networks**, often appear with concepts such as artificial intelligence, machine learning, and deep learning, which are closely connected but different.

Neural networks are often explained by comparing biological neural networks and a graph consisting of nodes and arrows. However, it is only an abstraction of the computation process, not the formal definition. Besides, it is neither persuading to draw a parallel between biological neural networks and artificial neural networks (Goldberg, 2017, p. 2; Koehn, 2020, p. 31). Guresen and Kayakutlu tried to define artificial neural networks in 2011 formally; however, the definition is quite complicated and redundant (Guresen & Kayakutlu, 2011). In the end, if we want to grasp the essentials, neural networks are connected and structured sets of linear and non-linear mathematical functions.

In a nutshell, neural networks are not equal to deep learning, machine learning, or artificial intelligence, but an approach to the latter, an approach to achieve machine translation. Neural machine translation models the task as a machine learning problem and uses neural networks to solve it.

2.2.2. Neural network basics

Although neural networks are far from as intricate as the human brain, it is easier to understand it thinking about how the brain works. A single neuron connects with other neurons. All neurons and their interconnections form a complicated network. The behavior of one neuron affects that of other neurons. But artificial neural networks are much simpler than biological ones: one or more **neurons** are arranged into a layer, and several layers are arranged into the network. The first layer is called the **input layer,** the last the **output layer,** while the rest are called **hidden layers**. Except for the input layer neurons, which only serve as memory storage of the input and pass the data into the network, all other neurons are mathematical functions that take inputs and transform them to a single output. For each neuron, it connects to certain neurons of the previous layer and the following layer.

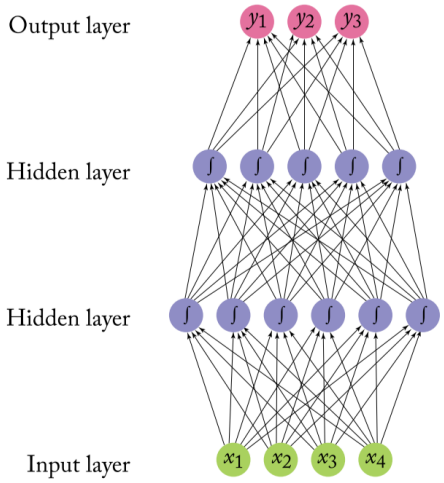

Figure 2.2 A typical representation of neural networks (Goldberg, 2017, p. 42).

For example, in Figure 2.2, the input neuron x_1 connects to all the six neurons of the following layer. Each connection is represented as an arrow. These connections are essentially numbers called **weights**[3]. When x_1 receives a real number, each connection multiplies the number according to its weight and passes the result to the neuron it connects in the following layer as one of its inputs. Then in the following layer, each neuron uses all the inputs it receives to output another real number. How it transforms the inputs depends on the function assigned to it, which is named **activation function**. There are four common activation functions used in neural networks:

1) Sigmoid function or logistic function
2) Tanh function or hyperbolic tangent function
3) ReLU function (ReLU: rectified linear activation unit)

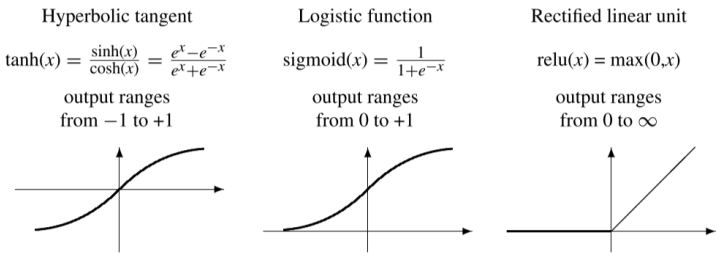

Figure 2.3 Equations and graphs of three common activation functions (Koehn, 2020, p. 70).

4) Softmax function: The softmax function usually is applied to the output layer that has only one node. Simply put, it takes in a *k*-length real value vector and turns it

[3] Often referred to as parameters in neural networks training

into another *k*-length vector where each value is between 0 and 1. The element-wise sum of the new vector equals 1, representing a probability distribution.

Note that neurons will first add up all the inputs they receive then use the sum as an independent variable to the function except for softmax. Besides, apart from the output layer, there is usually a bias neuron in each layer, which always equals 1 (Koehn, 2020, p. 71). The bias neuron only needs to connect to the next layer.

More abstractly, we can summarize the neural network as vector and matrix operations. For example, in the neural network architecture in Figure 2.2, we can view the input (plus the bias neuron) as a vector $\boldsymbol{x} = [x_1, x_2, x_3, x_4, 1]$. First it will be multiplied by a weight matrix $\boldsymbol{W}$ of shape (5,6) to yield a vector $\boldsymbol{h}' = [h_1', h_2', h_3', h_4', h_5', h_6']$. The vector $\boldsymbol{h}'$ is actually the summed input for each neuron in the first hidden layer. Then, we apply the activation function to each element of $\boldsymbol{h}'$ to get another vector of $\boldsymbol{h} = [h_1, h_2, h_3, h_4, h_5, h_6]$, multiplied by another weight matrix of shape (6, 5) for the second hidden layer. Furthermore, these operations can be done for several input sets simultaneously by stacking all input vectors into a matrix.

Once confirmed the number of layers, the number of neurons in each layer, and the activation function for each layer, we can start the training process. Typically, a neural network will start at a set of arbitrary weights. Then, it will compare the real output with the expected output for each training sample and adjust the weights till the training stops. During this process, there are two crucial questions: how should each weight be adjusted, and when should the training process stop, which we will cover in the following sections.

After training, the neural network can then be used to predict on new inputs, process which is referred to as **inference**.

2.2.3. Training: loss function, learning rate, and optimizer

Say we have a training sample of $(x_{1:n}, y_{1:m}), x_{1:n} = [x_1, x_2 \dots x_n], y_{1:m} = [y_1, y_2 \dots y_m]$. When we feed $x_{1:n}$ into the network at some point of our training, the output is $\hat{y}_{1:m} = [\hat{y}_1, \hat{y}_2 \dots \hat{y}_m]$. In order to know how to adjust the weights, we first need to compare the actual output $\hat{y}_{1:m}$ with the desired output $y_{1:m}$. There are also different functions to do this, for example, Hinge, log loss, and cross-entropy loss (Goldberg, 2017, pp. 26–28). We call them

loss function, cost function, or error function (Goodfellow et al., 2016, p. 82). The loss function **L** suitable for training varies depending on the characteristics of the specific task.

Generally speaking, a successful training would consist of finding a set of weights Θ that make the loss as small as possible over the whole training set. Suppose we perceive L as a function of Θ. In that case, we can transform that question into finding the minima of a function and get the independent variable' value at that point. Intuitively, it is about calculating $\boldsymbol{L}''(\Theta) = 0$; however, this method is complex when there are multiple independent variables. Instead, we turn to calculate the changing rate of L in terms of each element of Θ (essentially its partial derivatives) and slowly adjust Θ in proportion to that rate against the changing direction in the hope of reaching a desired set of weights (Goldberg, 2017, pp. 30–31). This type of method is called gradient[4]-based optimization. See a detailed calculation sample written by Koehn (2020, pp. 73–77). During training, the proportion to the changing rate is a hyperparameter[5] called the **learning rate**. Usually, we want to keep the learning rate small so that the change will not be too drastic to go over the minima point. Ideally, we adjust the weights for every training sample passed into the neural networks, known as **SGD (stochastic gradient descent)** algorithm.

Note that there are many points where the gradient is zero, but the loss function is not at its lowest point. We refer to this point as **local minima**. It is usually problematic for neural network training as the learning can be stuck at local minima without finding the **global minima** on the whole loss function.

Researchers have invented several other algorithms to reduce training time and avoid local minima (Goldberg, 2017, p. 35). These algorithms are usually referred to as **optimizers** in machine learning toolkits.

1) Mini-batch SGD: adjust weights every *n* training samples to reduce noise in a single training sample.
2) SGD+Momentum and Nesterov Momentum: current gradient is affected by its previous gradient.

[4] Gradient is a mathematical concept, essentially a vector of all partial derivatives of a function with multiple independent variables.
[5] Parameters that are set manually before training starts instead of learned (Mehryar Mohri et al., 2018, p. 4).

3) AdaGrad, AdaDelta, RMSProp, and Adam: adaptive learning rate algorithms that change learning rate per minibatch or weight (Ruder, 2017).

An advantage of adaptive learning rate algorithms is that we do not need to select a learning rate. The training result is usually good when sticking to the default value (Ruder, 2017). Mainly, Adam has been selected as the default choice in many scenarios due to its efficiency and robustness (Goldberg, 2017, p. 59; Ruder, 2017).

2.2.4. Training: exploding and vanishing gradients

Gradient-based optimization has a significant downside of exploding and vanishing gradients. The weight update bases on the gradient, essentially a multiplication using outputs and weights of later neurons (Koehn, 2020, pp. 73–77, 183). These are values that lie in the range of [–1,1] or greater than one, as indicated in Figure 2.4. especially. Hence, the multiplication tends to yield either a nearly zero or huge value for networks with lots of layers. The former is called the vanishing gradient, while the latter the exploding gradient. The vanishing gradient nearly does not update weight and makes the neurons unable to learn from new input. The exploding gradient makes the loss unstable, posing more difficulties when training (Koehn, 2020, pp. 113, 173).

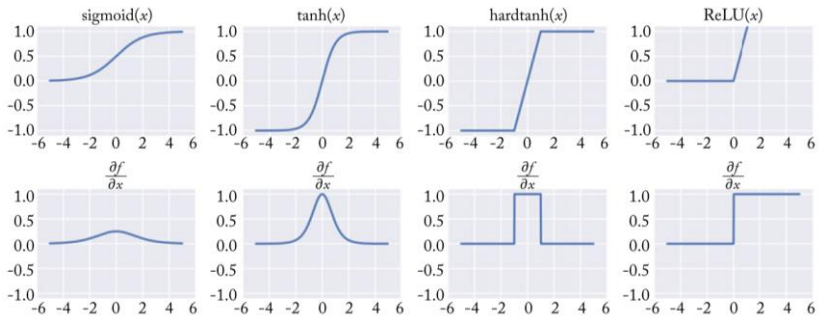

Figure 2.4 Graphs of activation functions and their derivative (Goldberg, 2017, p. 46).

Koehn, Goldberg, and Goodfellow *et al.* all mentioned techniques to address vanishing and exploding gradients, including data normalization, gradient clipping, and network structure adjustment (Goldberg, 2017, pp. 60–61; Goodfellow et al., 2016, pp. 413–416; Koehn, 2020, pp. 182–185). They can be summarized as below:

1) Gradient clipping
    a. Element-wise minibatch clipping
    b. Norm clipping

c. Adaptive gradient clipping

2) Data normalization
   a. Batch normalization
   b. Layer normalization
3) Network structure adjustment
   a. Residual connection or skip connection
   b. Highway network

2.2.5. Regularization: generalization problem

Neural networks are powerful in finding patterns in big data sets to make predictions for unseen inputs of the same task (Goodfellow et al., 2016, p. 110). A significant factor influencing neural network performance is how well it learns about training data.

There are two imperfect scenarios: underfitting and overfitting. Underfitting happens when the number of parameters used in a model is insufficient to define a complex task and gets a relatively large loss. Similarly, overfitting can happen when there are too many parameters. Even though the model performs well on training data, it does not fit with unseen data (Koehn, 2020, p. 147). It is called the generalization problem within the field of machine learning. Koehn uses a great illustration in his book to explain this problem.

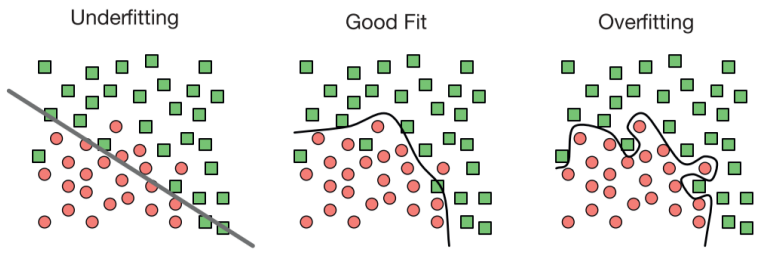

Figure 2.5 Illustration of underfitting and overfitting (Koehn, 2020, p. 147).

More often than not, neural networks have parameters counted in the order of millions, so we mainly need to worry about overfitting rather than underfitting. There are also various techniques addressing the overfitting issue (Goldberg, 2017, p. 47):

1) Regularizers: L_2, L_1, and elastic-net
2) Dropout training
3) Early stopping (Mehryar Mohri et al., 2018, p. 165)

Regularizers use the parameter values and calculate their **complexity (generalization loss)**. Then, the ultimate training goal is to keep both the training loss and generalization loss low. The dropout training is another technique that helps the model not to rely on seen data. It randomly sets the value of a certain quantity of neurons to zero for a single training sample in SGD or for a mini-batch, in which way their parameters will not be updated. The dropout neurons vary per training sample or mini-batch (Goldberg, 2017, p. 47; Koehn, 2020, pp. 181–182).

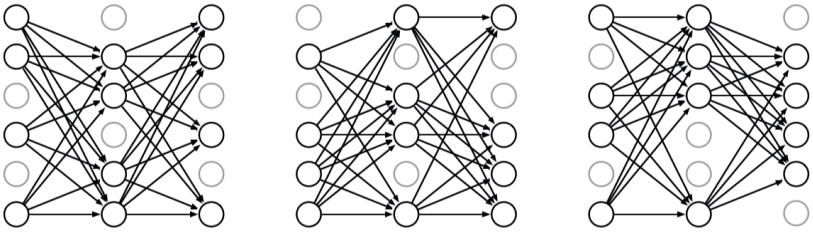

Figure 2.6 Illustration of dropout training (Koehn, 2020, p. 182).

There are also other techniques to smooth the training process. For example, "warmup steps" represents a technique to slowly increase the learning rate from zero till the configured number. It helps to avoid failed training due to the network's sudden exposure to differentiated data (Popel & Bojar, 2018). "Weight decay" is used with regularizers mentioned to prevent overfitting (Vasani, 2019). A reasonable weight decay would be between 0 and 0.1 (Kuhn & Johnson, 2013, p. 144). We will see these hyperparameters in Section 3.3.8.

2.2.6. Early stopping: train, validation, and test sets

As for now, we have discussed how to adjust the weights or, more technically speaking, optimize the neural network so that we slowly reach the desired set of weights. However, neural networks overfit over many rounds on the whole data sample (an **epoch**[6]). The question of deciding when to stop training stays unresolved.

This issue is addressed by splitting the whole training dataset into two sets, one used for training (**train set**) and the other for evaluating the generalization ability of the current model (**validation set**). For example, each time after feeding one epoch into the neural network,

[6] In neural network training, each round it passes through the whole dataset is called an epoch, sometimes also referred to as batch. That is the reason why a small number of training sample is called mini-batch. (Koehn, 2020, pp. 73, 80)

we can use the validation set to calculate the loss. Usually, the training loss will continuously decrease as training progresses, but the validation loss will first decrease then increase at a certain point (Goodfellow et al., 2016, pp. 245–249). That turning point is when we should stop training the model, referred to as **early stopping**. See the illustration below:

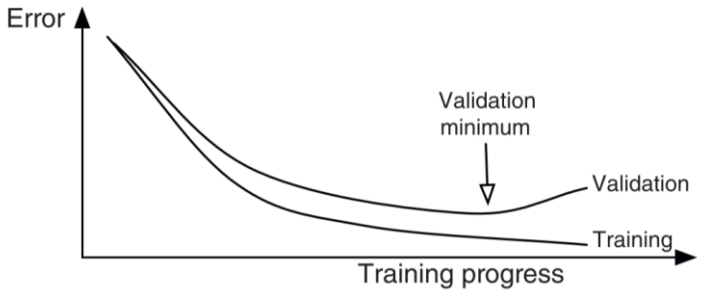

Figure 2.7 Change pattern of training loss and validation error as training progresses (Koehn, 2020, p. 79).

There are two major methods of early stopping (Allibhai, 2018; Goldberg, 2017, pp. 14–16):

1) Hold-out: directly split the training set into two subsets based on predetermined percentages, for example, 80%/20%.
2) Cross-validation: split the training set into *k* subsets; train the model *k* times; each time use one subset for evaluation and the rest as training sets; calculate the average generalization loss on *k* split groups.

However, we usually want to train several models adjusting hyperparameters in real scenarios and then select the best model. It requires further splitting the whole dataset into three parts: train, validation, and **test sets**. Each time adjusting hyperparameters, we use the train set and the validation set to get the best weights for that model. After getting several models, we use the test set to evaluate their performances on new inputs to select the best model (Goldberg, 2017, pp. 15–16).

2.3. Neural Machine Translation

As presented in the beginning of this chapter, neural machine translation is an approach to language conversion automation using neural networks. Though most of the principles of neural networks apply to it as well, there are several particularities unresolved.

2.3.1. Word embeddings

Neural networks take real numbers as input, but the initial input and final output for NMT models are words. The first problem we need to resolve is how we convert words into numbers. At the same time, words are not atomic items, meaning similarities could be drawn among their mathematical representations.

But before we dive into explaining word embeddings, it is necessary to introduce first the concept of **token** and **tokenization** in the field of natural language processing. Manning *et al.* loosely metaphorize the tokens as pieces chopped off a sentence (Manning et al., 2008, p. 21). Moreover, this fragmentation is not a random process but aims to segment the text sequence into "distinct meaningful units (or tokens) before any language processing beyond the character level can be performed" (Kaplan, 2005). Hence, tokenization is the process of segmenting a textual sequence into small meaningful units, which are referred to as tokens. Manning *et al.* also distinguish **type** from token. Type is a fixed pattern of characters while token is an instance of a type (Manning et al., 2008, p. 21). Their relationship is similar to that of vocabulary and words.

Seemingly easy, however, tokenization requires further considerations. From the perspective of writing systems for English and European languages, this convention is intuitive as there are obvious blank spaces between two words. However, it is not the case for Chinese or Japanese where the equivalents of "words" are not separated by more space than what separates their version of "morphemes" or "syllables". In fact, even in English this phenomenon happens if you think of "notwithstanding" (Manning et al., 2008, p. 25).

Neural machine translation works on types and tokens. A hypothesis is that there is a *N*-dimensional space, where *N* is minor to the number of types in certain language, that "is sufficient to encode all semantics of our language" (Francois Chaubard et al., 2019). It is essentially what has been done. Researchers first encode all types into one-hot vectors[7] of the same dimensionality as the number of types, then use different methods to reduce the dimensionality.

[7] A vector with only one element with the value of 1 and the rest with value of 0.

There are purely mathematical methods to achieve this goal such as singular value decomposition (SVD), as well as neural network approaches (Francois Chaubard et al., 2019). If we design a neural network in order to predict the next word given a sequence, the number of neurons in the first layer should be the same as the number of types (often simply referred to as vocabulary in many cases) plus one bias neuron. Then we can set the first hidden layer to have 512 neurons for example. After training, each neuron of the input layer will have ordered 512 weights connected to the first hidden layer. When a token enters the input layer, only one neuron will be activated and has the value of 1. Thus, what enters the neural network and represents the token is those 512 weights in order, which are essentially the **word embedding** for that token/type. The task of predicting the next word mentioned above is called **language modeling**, a common term and task of natural language processing.

Word embeddings are then essentially a dense vector[8] of reduced dimensionality compared to one-hot vectors. Words that have similar semantic information also have similar word embeddings due to the fact that related words often appear in the same context and more likely to be followed by the same word, which obliges to network to use similar information to encode related words. Otherwise, the network may yield an animal for the input "My favorite fruit is".

Google has published a web application to visualize word embeddings (available at: https://projector.tensorflow.org/) (Smilkov et al., 2016). From the example below one can see that the nearest points to "hours" in the original dimensional space are "minutes", "hour", "weeks", etc. More interestingly, you can even do mathematical operations on word embeddings vectors (Koehn, 2020, p. 108; Mikolov et al., 2013).

[8] A vector with many non-zero values as opposed to one-hot vector.

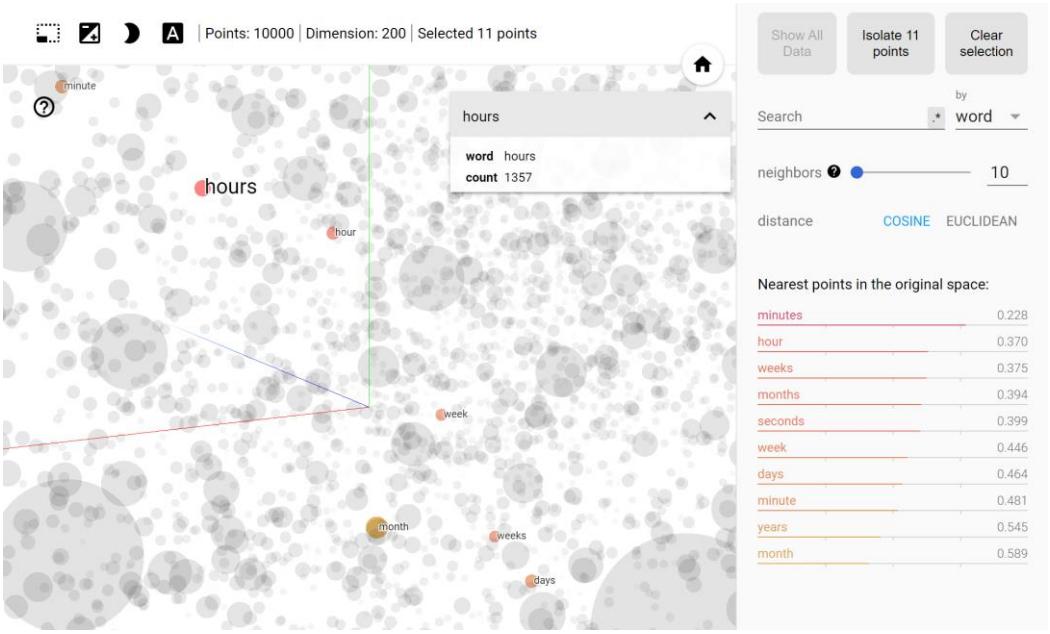

Figure 2.8 Visualization of word embeddings from Embedding Projector.

Word embeddings are actually a side product of language modeling as they are not the ultimate goal of this task. As neural machine translation also works by predicting the next output in the target language, the weights between the input layer and the first hidden layer can still be considered as word embeddings. In fact, this hidden layer is simply referred as the **embedding layer**.

However, word embedding cannot solve polysemy problem as they are fixed once training is done (Koehn, 2020, pp. 218–219) and largely influenced by how the words are represented in the training corpus. For example, the system will not understand "bank" as geographical term if it has been trained mostly on financial texts where the "bank" is embedded as a financial institution.

2.3.2. RNN, LSTM and GRN

Another problem for applying neural networks to machine translation is that the input and output length are not fixed as the sentences vary in length as well. This is where **recurrent neural networks** (RNNs) come in. RNN takes in one token x_{t-1} at a time to predict the next output token $\vec{y}_{t-1}$. The hidden state h_{t-1} of x_{t-1} will be saved and passed onto the next processing time step in order to calculate the hidden state h_t of x_t and then predict the following output token $\vec{y}_t$. See the illustration graph of this process below.

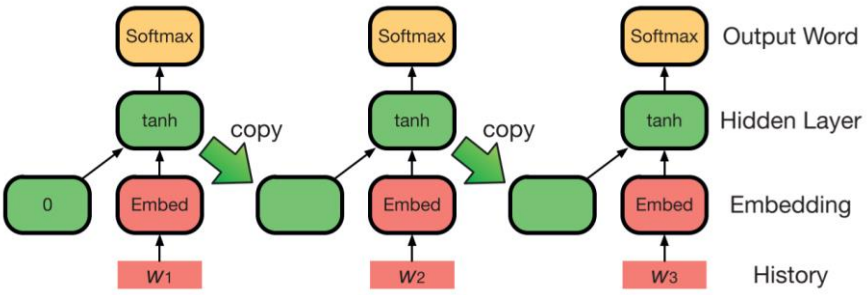

Figure 2.9 Illustration graph of RNN processing details (Koehn, 2020, p. 110).

In this way, inputs of any length can be passed into the RNN; however, it also has certain disadvantages, for example, the network tends to "forget" the initial words if the input sequence is very long (Koehn, 2020, p. 112). To tackle this problem, RNN with long short-term memory (LSTM) or gated recurrent units (GRUs) is invented so that the neural network can handle long-distance dependencies. Besides, RNNs are also more time-consuming and require greater computing power (Huang et al., 2013; Ouyang et al., 2017).

Even though RNN, LSTM, and GRU are more complicated in structure and computation process, they apply the same training philosophy as normal neural networks that includes training loss, learning rate, optimizer, regularizer, gradient descent, etc.

As the transformer is the focus of this thesis, which has established its position as state-of-the-art and is applied widely in all tasks, we will not delve into the details of RNN. However, RNN did have its time for the workaround in dynamic sequence length, and the ability to attend to long-distance dependencies.

2.3.3. Encoder-decoder hyper-architecture

Neural networks applied to translation task commonly consist of two hyper structures: the encoder and the decoder. Essentially, the encoder encodes the source text into a final hidden state, which then is used with target inputs to predict the output. See an example of a RNN encoder-decoder framework below:

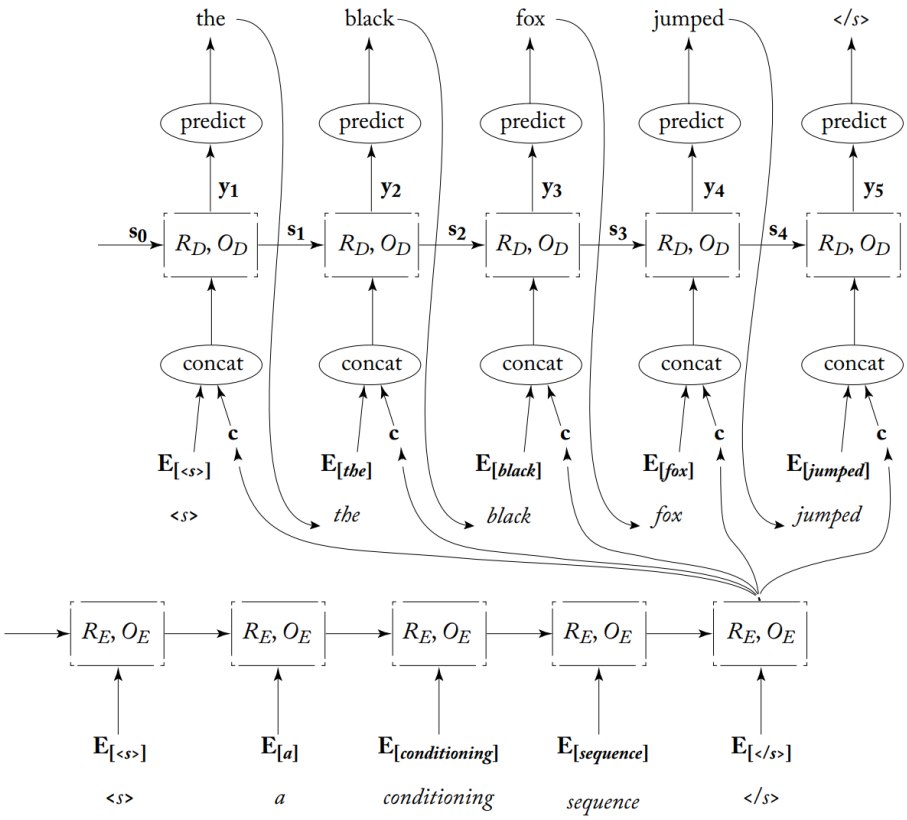

Figure 2.10 Example of encoder-decoder framework for predict a sequence on another given sequence (Goldberg, 2017, p. 199).

The RNN takes in a sequence of information and generates another sequence of information, both variable in length. This is the reason why machine translation models in the field of deep learning are often referred to as sequence-to-sequence model, which also includes tasks such as question answering, text summarization, voice to text, etc.

2.3.4. Beam search

When training a machine translation neural network, the decoder would output one word at a time. Naturally, the trained model would also predict the output word by word choosing the one with the highest probability after the softmax function; however, it is possible that the best translation does not start with the word predicted as the most probable. In fact, the model preserves various options ranked by probability at each time step. The list of words is called a **beam**. Then, another beam is generated for each word of the anterior beam till the prediction indicates the end of the output sequence.

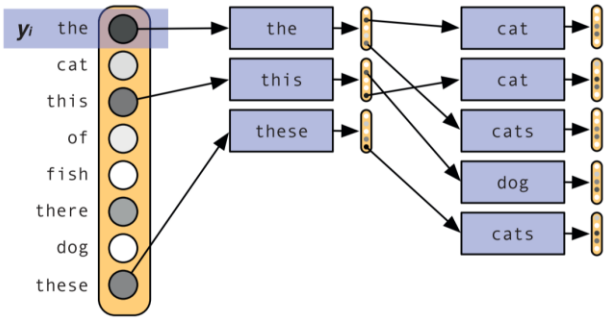

Figure 2.11 Example of beam prediction of neural machine translation model (Koehn, 2020, p. 146).

After all predictions are performed, the model then chooses the best translation considering all possible routes along these words, which is the route that yields the highest probability. This whole decoding process is referred to as **inference** in machine learning.

2.4. Transformer

On Dec. 5, 2017, Vaswani *et al.* from Google published a paper titled *Attention Is All You Need*, in which the research team abandoned recurrence for seq2seq[9] task and proposed self-attention as its substitution (Vaswani et al., 2017). The model named Transformer since became state-of-the-art in many NLP tasks, including machine translation. Below is the original abstraction of the model design.

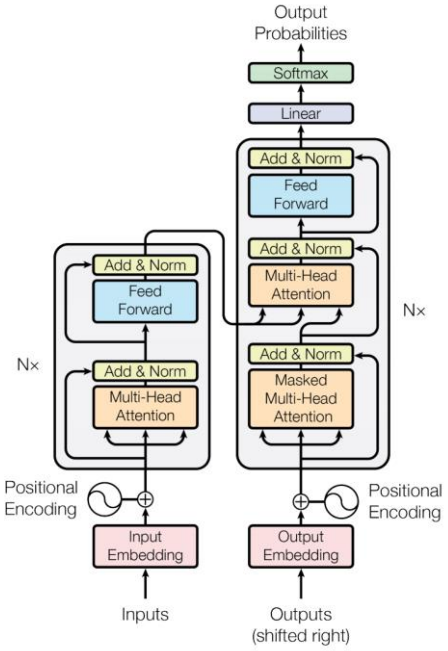

Figure 2.12 Illustration of the original Transformer (Vaswani et al., 2017)

[9] Sequence-to-sequence

On the left is the encoder, on the right is the decoder. Suppose we imagine the encoder as a layer. In that case, the layer can then be divided into two sublayers: the multi-head attention sublayer and the feed-forward sublayer, each with an add & normalization operation on the output. The "Nx" means that there are N layers linearly connected. The final output of the previous layer will be fed into the next layer as input. Similarly, the decoder can then be divided into three sublayers: masked multi-head sublayer, multi-head attention sublayer, and the feed-forward sublayer.

In Transformer, the model processes all words at once (except for the decoder during inference), which is its key difference from recursive neural networks. Hence, Transformer "allows for significantly more parallelization" (Vaswani et al., 2017).

2.4.1. Input embedding

As explained in Section 2.3.1, Transformer also uses a one-hot encoding vector for words and using a feed-forward layer to obtain the word embeddings. In the original paper, the Google research team set the length of word embedding vectors to d_{model} = 512 (Vaswani et al., 2017). We will maintain this dimensionality to explain the Transformer.

2.4.2. Positional encoding

As the Transformer process all words in the input sequence simultaneously, it does not have any sense of word order. However, this information is critical to language understanding. To tackle this problem, Vaswani *et al.*incorporate positional encoding to word embeddings (Vaswani et al., 2017).

They used two equations to calculate each element of the positional encoding of the word x_{pos} in an input sequence [x_1, x_2… x_n]:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$i \in [0, 1, 2 \ldots \frac{d_{model}}{2} - 1]$$

The positional encoding of the word x_{pos} then is:

$$PE_{pos} = [PE_{(pos,0)}, PE_{(pos,1)}, PE_{(pos,2)} \dots PE_{(pos,d_{model}-1)}]$$

There are various reasons why they encode this way, but we will not delve into its mathematical details. For positional encoding, we only need to understand the following points (Vaswani et al., 2017):

1) Each positional encoding is of the same length as word embedding. Both equal to d_{model}.
2) For a specific d_{model}, all the x_{pos} in different input sequences will have the same positional encoding. For example, for sequence *A* = "This is an attention" and *B* = "All you need is attention", $PE_{(A, 2)}$ (positional encoding of *an* in *A*) is the same as $PE_{(B, 2)}$ (positional encoding of *need* in *B*).
3) In real scenarios, all positional encodings of the words in a given input sequence are different, which means that this method is valid to represent the position information.

For word x_{pos} in a sequence $[x_1, x_2 \dots x_n]$, its word embedding vector $E_{pos} = [E_{(pos, 0)}, E_{(pos, 1)}, E_{(pos, 2)} \dots E_{(pos, 511)}]$ and its positional encoding vector $PE_{pos} = [PE_{(pos, 0)}, PE_{(pos, 1)}, PE_{(pos, 2)} \dots PE_{(pos, 511)}]$ will be summed to obtain the final input vector $I_{pos} = [E_{(pos, 0)} + PE_{(pos, 0)}, E_{(pos, 1)} + PE_{(pos, 1)} \dots E_{(pos, 511)} + PE_{(pos, 511)}]$.

2.4.3. Self-attention

Like the attention mechanism in RNN-based encoder-decoder, self-attention also focuses on different parts of a sequence given a condition. However, in the encoder part of the Transformer, this condition is an element of the sequence itself, which is why it is named self-attention.

In the last section, we have obtained the input vector $I_{pos} = [I_{(pos, 0)}, I_{(pos, 1)}, I_{(pos, 2)} \dots I_{(pos, 511)}]$. When it enters one "head" of the multi-head attention sublayer, it will be multiplied by three different matrices *Q*, *K*, and *V* to get vectors q_{pos}, k_{pos}, and v_{pos}. q_{pos} and k_{pos} have length of d_k while v_{pos} has length of d_v. Hence, *Q* and *K* have shape (d_{model}, d_k) and *V* shape (d_{model}, d_v). This process can be illustrated as below:

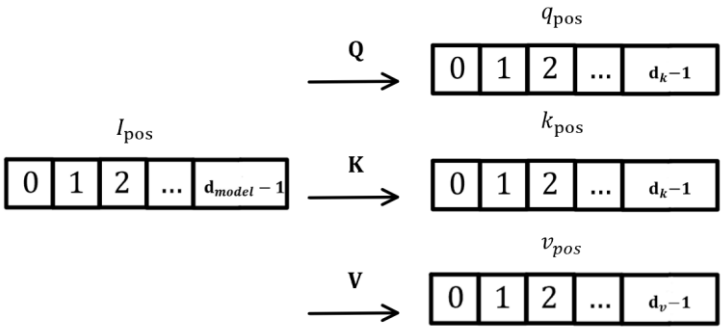

Figure 2.13 Query, key, value vectors generation process

The letters *Q*, *K*, and *V* represent query, key, and value from retrieval systems. Note that matrices *Q*, *K*, and *V* are learned during training and will be the same for all the input vectors $I_1, I_2 \dots I_n$.

After processing all the input vectors, we get three series of vectors:

1) $q_1, q_2, q_3 \dots q_n$
2) $k_1, k_2, k_3 \dots k_n$
3) $v_1, v_2, v_3 \dots v_n$

For position 1:

1) Calculate $q_1 \cdot k_1, q_1 \cdot k_2 \dots q_1 \cdot k_n$ respectively. Note that it is a dot product operation between two vectors of the same length that returns a single number (also called scalar in Linear Algebra). The result will be $\widehat{w}_1^1, \widehat{w}_2^1 \dots \widehat{w}_n^1$ (the letter *w* represents weight). Intuitively, we can interpret these weights as relationship intensity between position one and all the positions in the sequence.
2) The original Transformer model divides each weight by $\sqrt{d_k}$ and then pass the results into a softmax function to get a probability distribution of new weights $w_1^1, w_2^1 \dots w_n^1$. Note that $1 = w_1^1 + w_2^1 + \cdots + w_n^1$.
3) Then, we do $w_1^1 v_1 + w_2^1 v_2 + \cdots + w_n^1 v_n$ and yield a vector $\hat{z}_1$ of length d_v, which will be used for the add & normalization operation.

If we stack respectively all the q_{pos}, k_{pos}, and v_{pos} vectors into matrices, the whole calculation until the probability distribution of weights on all positions is (Alammar, 2018; Vaswani et al., 2017):

$$\begin{bmatrix} q_{(1,0)} & \cdots & q_{(1,d_k-1)} \\ \vdots & \ddots & \vdots \\ q_{(n,0)} & \cdots & q_{(n,d_k-1)} \end{bmatrix} \times \begin{bmatrix} k_{(1,0)} & \cdots & k_{(1,d_k-1)} \\ \vdots & \ddots & \vdots \\ k_{(n,0)} & \cdots & k_{(n,d_k-1)} \end{bmatrix}^T \xrightarrow{\frac{1}{\sqrt{d_k}}}$$

$$\xrightarrow{Softmax} \begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_{n-1}^1 & w_n^1 \\ w_1^2 & w_2^2 & \cdots & w_{n-1}^2 & w_n^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_1^{n-1} & w_2^{n-1} & \cdots & w_{n-1}^{n-1} & w_n^{n-1} \\ w_1^n & w_2^n & \cdots & w_{n-1}^n & w_n^n \end{bmatrix}$$

The table below is a direct representation of this operation. Note that we apply the softmax function to rows, not columns.

	Hi	how	are	you
Hi	0.7	0.1	0.1	0.1
how	0.1	0.6	0.2	0.1
are	0.1	0.3	0.6	0.1
you	0.1	0.3	0.3	0.3

Figure 2.14 Self-attention scores for a sequence. Adapted from (Phi, 2020).

There is also a powerful visualization tool, BertViz, for interpreting attention in the Transformer model (Vig, 2019), available at https://github.com/jessevig/bertviz.

2.4.4. Multi-head attention and normalization

As explained above, one head of attention yields n vectors $\hat{z}_{pos}$ of length d_v, but the Transformer is more than that. It uses $h = 8$ heads originally. Besides, Vaswani *et al.* set $d_k = d_v = d_{model}/h = 64$. Each head uses a different set of Q, K, and V and ideally learns distinct features. If we denote the output vector of head one as $\hat{z}_{pos}^1$, the calculation of the final h-heads attention for position one is:

1) Concatenate orderly all single-head attention vectors into one single vector.
2) Multiply this single vector by a matrix W^O of shape (hd_v, d_{model}) to yield the final attention vector z_1 of length d_{model} for position one.

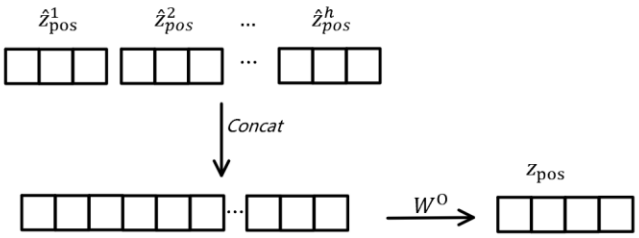

Figure 2.15 Concatenation of single-head attention vectors and yielding the final attention vector

For the add & normalization operation:

1) Add input vector I_{pos} and final attention vector z_{pos}.
2) Normalize the values of the vector addition product and send it to the feed-forward sublayer.

The original Transformer uses layer normalization proposed by Ba *et al.* in order to tackle the vanishing gradient problem and reduce training time (Ba et al., 2016; Goldberg, 2017, p. 60). It could be viewed as a refinement of the model to make it perform better.

2.4.5. Encoder and decoder in Transformer

Since all sublayers in the Transformer produce vectors of d_{model} dimensions, the encoder and decoder can be stacked respectively for N times, as the model architecture illustrates. The final output of the whole encoder part is n vectors of length d_{model}, which can be viewed as a matrix of shape (n, d_{model}).

Like the self-attention mechanism, these vectors will first be transformed by another set of K and V matrices into another two series of k_{pos} and v_{pos}, then used in the middle sublayer of the decoder layer. Note that the length of the input sequence to the decoder, say m, is different from that of the encoder, but it has no influence on operations.

The decoder is slightly different from the encoder: one layer in the decoder has three sublayers. The final decoder outputs (m vectors of length d_{model}) connect to a linear classifier layer, which has the same number of neurons as the vocabulary size. Note that the final decoder output is a matrix of shape (m, d_{model}); however, the linear classifier only expects a vector input. Actually, the linear classifier uses the vector in the position m to calculate and then pass them to the softmax function to generate the possibility distribution on the whole vocabulary.

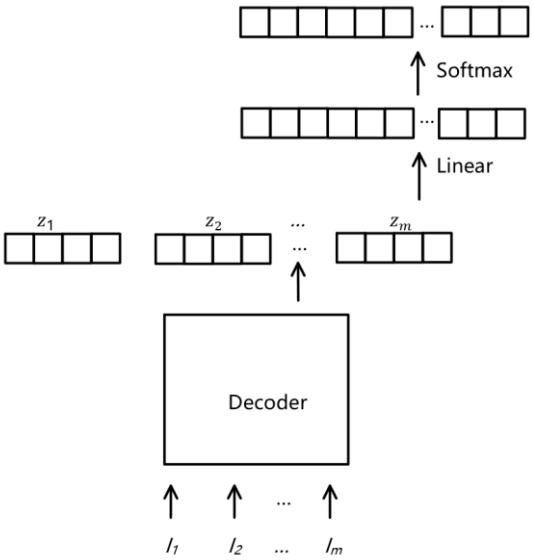

Figure 2.16 Simplified decoder prediction process

Then, during training, this possibility distribution can be compared against the desired output in the training sample to yield loss function or to be used to predict the next word during inference.

2.4.6. Masked attention

Another detail we have not touched is that the bottom sublayer of the decoder layer uses the variance of the self-attention applied in the encoder. The intention is simple: when we are training the Transformer, the decoder receives all the words of the target sentence; however, it should not have access to the information after the position it is calculating. For example, if the decoder receives "<start> I am fine" as input, when predicting the word in position 3, which is "am" in this case, it should not attend to positions 3 and 4.

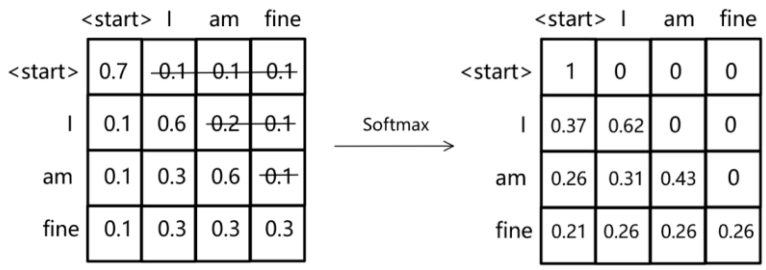

Figure 2.17 Masked attention. Adapted from (Phi, 2020).

The masked attention eliminates the values of future positions and re-apply the softmax function to the query matrix (Alammar, 2018). This way, the decoder can take all the input sentences in unison and do matrix operations to reduce training time. It will not need to process 3 conditioned predictions only one at a time, which are P (position$_2$ | <start>), P (position$_3$ | <start> I), and P (position$_4$ | <start> I am).

2.5. Machine Translation Evaluation

However evolutionary a machine translation approach is, its output quality evaluation can still depend on how translation is commonly evaluated through human judgment via certain methodology, analyzing grammatical, semantic, and stylistic aspects (Poibeau, 2017, p. 132), addressing issues such as its accuracy, fluency, error typology, etc. But there are also automatic strategies invented along the way to reduce the high cost of human intervention (Papineni et al., 2002; Turian et al., 2006) in the quality evaluation process.

These strategies are evaluation metrics such as BLEU, NIST, and METEOR. Conceived in 2002, BLEU is likely the most famous and widely-used metrics nowadays, with many competitions uses this score as benchmark criteria. BLEU requires at least one human translation sentence as reference and compares the similarity of the reference and the hypothesis translation proposed by machine using N-grams. It also takes the length ratio into account in order not to favor too short sentences. BLEU score ranges from 0 to 1, with 1 as the exact same as the human translation. Quite often than not, one can also find that BLEU score is represented by a range of 100. However, BLEU is designed to be only interpretable on a corpus level as it often disagrees with human judgment on individual sentences (Papineni et al., 2002).

NIST is similar to BLEU in that it compares the sentence similarity as well, but NIST favors rarer segments. And METEOR is yet another trying improvement on BLEU. Banerjee and Lavie, authors of METEOR, report that this metric yields closer judgement result to human evaluation (Banerjee & Lavie, 2005); however, this method is also more complicated to apply and interpret, hence less frequently used (Poibeau, 2017, p. 136).

These metrics may correlate well with human judgement (Banerjee & Lavie, 2005; Papineni et al., 2002), but there are also studies argue that they can easily disagree with human

evaluation results (Callison-Burch et al., 2006) and are overly relied upon (Mathur et al., 2020).

3. Methodology

After diving into various concepts and details about neural machine translation, we were finally training our models. The actual training and coding process was far from as complicated as the principles and theories; however, they were necessary to understand what different functions and methods mean in training toolkits and helped to read and twist the code.

This chapter describes how we constructed a custom parallel corpus and used it to fine-tune (see Section 3.3.1) an NMT model. The experiment was a practical application of the theories explained in the last chapter.

3.1. Objectives

The experiment aimed to fine-tune a general Spanish to Simplified Chinese translation model to the legal domain, a process which served both as a hands-on tutorial of *transformers* library usage and as a practical application of neural machine translation theories explained in the Theoretical Framework chapter.

Besides this, we also wanted to gain insights about machine translation performance enhancement, domain-specific fine-tuning challenges, and suggestions on following experiments, particularly for the language pair of Spanish and Simplified Chinese.

The decision to fine-tune a pre-trained model instead of training a new one from scratch was conditioned by bitext scarcity between our language pair in the legal domain. Also, training from scratch on a much larger corpus necessitate a considerable amount of time and computations, which is not practical for a master thesis, especially when there are hardware limitations.

The experiment had three major stages: bitext construction, training, evaluation. Each stage then was divided into specific tasks or processes.

Also, we created a GitHub repository to grant open access to relative materials such as processing codes and output files for the public. We recommend that readers not familiar with Hugging Face or Python read this chapter with the help of those materials. The repository is available at: https://github.com/guocheng98/MUTTT2020_TFM_ZGC.

3.2. Bitext Construction

This stage leveraged several legal domain databases and some other Internet resources to find Spanish and Simplified Chinese texts, which were later aligned manually in SDL Trados Studio 2019. Various files, mainly Spanish, have been uploaded to the GitHub repository.

3.2.1. Resources used

The only available corpus for Spanish and Simplified Chinese in an approximate domain is the United Nations Parallel Corpus (UNPC). Nevertheless, it also contains many other non-legal documents such as meeting briefings, reports, statements, and communications, as the Official Document System of the United Nations shows[10]. We did not use UNPC because it would be challenging to select in-domain parallel segments.

Considering that fine-tuning requires less data, it is viable to construct a parallel corpus for the training manually. However, we found that there were barely bilingual crawlable Internet resources of our interest during the research, which meant that we would have to collect corresponding documents from both linguistic sides and manually align them to avoid low-quality alignment. It would be a one-way translation model from Spanish to Chinese, so it should be better to find available Chinese translations of Spanish legal documents firstly.

Professor Deng Pan from the China University of Political Science and Law has been translating various law codes of Hispanic countries over the years. One primary resource that we used was his co-translational work of Spanish Civil Code (Spanish Parliament, 2013), available on *Chaoxing (Huiya) Dianzi Tushu Shujuku 超星（汇雅）电子图书数据库*[11] [Chaoxing (Huiya) Digital Books Database] as print PDF format. We used the free version of *Tianruo 天若OCR*[12], an open-source optical character recognition tool that offers a free API for OCR developed by Baidu, Inc., to digitize the print PDF file into plain text.

[10] https://documents.un.org/prod/ods.nsf/home.xsp
[11] Requires database login. Available at: http://www.sslibrary.com/
[12] https://ocr.tianruo.net/

Another database used to find Chinese translation of Spanish legal documents was *Beida Fabao 北大法宝*[13] [China Law Info], which records Chinese laws, regulations, and cases in Chinese or bilingual version of English and Chinese. Specifically, the *Zhongwai Tiaoyue 中外条约* [Sino-foreign Treaties] and *Waiguo Fagui 外国法规* [Foreign Regulations] parts of the *Falv Fagui 法律法规* [Law and Regulations] section were used.

Figure 3.1 Parts and sections used from the China Law Info database.

In these two sections, we used the advanced search function to obtain treaties between the People's Republic of China and the Kingdom of Spain and all available European organizations in the hope of quickly finding the original text on EUR-Lex later. We batch-downloaded all results as plain text and did an initial selection judging from the titles of downloaded documents to filter out contents that were not closely related to the legal domain.

Also, we found a published Chinese translation[14] of the Spanish Constitution on *Mingde Gongfa Wang 明德公法网*[15], a platform of legal research news and materials created by the Renmin University of China, a leading academic institution in the legal domain.

The original Spanish text of the contents mentioned above was searched mainly on *Boletín Oficial del Estado*[16] (BOE) and EUR-Lex[17]. A tentative Spanish or English translation of a document's title was used to search in these platforms. After roughly comparing the content of possible results, we then decided if the Spanish document was the source text of that

[13] Requires database login for certain functions. Available at: https://pkulaw.com/
[14] http://www.calaw.cn/article/default.asp?id=4068, accessed on 21st June, 2021.
[15] http://www.calaw.cn
[16] https://www.boe.es/
[17] https://eur-lex.europa.eu/homepage.html

Chinese translation, hence creating an alignment document pair. We saved all Spanish texts as TXT files.

Spanish and Chinese documents of matching content had the same file name and were saved separately in two main folders. We included certain files in the GitHub repository, but not with the Chinese translations of the Spanish Civil Code and from the China Law Info database due to intellectual property rights protection.

3.2.2. Spanish Civil Code

As mentioned above, we used *Tianruo OCR* to transform the Chinese translation of the Spanish Civil Code into plain text. With the tool open, one can press *F4* to activate screen capture, select the area where the text is to be identified and wait for a few seconds for the tool to transform it into the program window. As the PDF file visualization was clear, the OCR tool barely made mistakes during the whole process. Besides, the transformed text generally maintained the original textual structure, separated correctly as paragraphs.

However, a few issues regarding line breaks did exist. As Figure 3.2 shows below, punctuation was missing at times, resulting in erroneous paragraph separation. Occasionally, there would be no separation between the article number and the content, as the second red box circled on both sides of the figure shows. This missing separation also happened when it came to chapter and section titles.

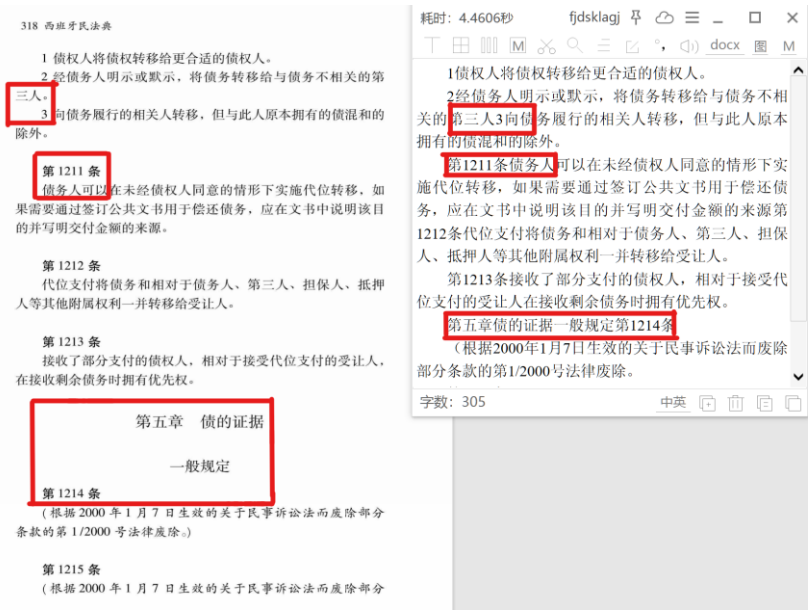

Figure 3.2 Program window and usage example of Tiranruo OCR.

These structural issues could prevent the CAT Tools from correctly segmenting sentences and paragraphs and further complicate the alignment procedure; hence, we manually corrected them in the OCR tool window, then copied the adjusted text into a TXT file.

The Civil Code contains four volumes (*libro* in Spanish), each volume divided by titles (*título*) apart from the preliminary title and final dispositions. We applied this structure to the TXT files. We saved each title in a separate file, with other titles in a folder representing their belonging volume.

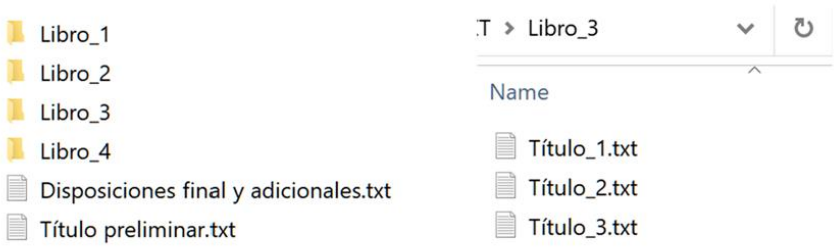

Figure 3.3 File structure of the digitized content of Spanish Civil Code.

After collecting the Chinese translation part, it was time to obtain the Spanish source text from the *Boletín Oficial del Estado*. Pan & Ma's translation had 17/10/2012 as the defining date for source text changes, so we chose the published version of 14/11/2012, the nearest available version to the one above.

Due to the relatively large quantity of text data, we decided to manipulate them directly from a locally saved HTML file using Python and Beautiful Soup library.

We first downloaded the webpage[18] as an HTML file, parsed the file into a Beautiful Soup object. As all information we needed was stored inside three HTML tags, namely "h4", "h5", and "p", we filtered these tags, extracted textual information from them, and then store them in a plain text file "spanish_raw.txt".

Later, we cleaned the raw text for that it had some apparent issues. Some numbered items were formatted as "1.ª" while some remained as "1.a". There were also many extra white

[18] https://www.boe.es/buscar/act.php?id=BOE-A-1889-4763&p=20121114&tn=0

spaces between words. After these simple cleaning processes, we stored the new text into "spanish_cleaned.txt".

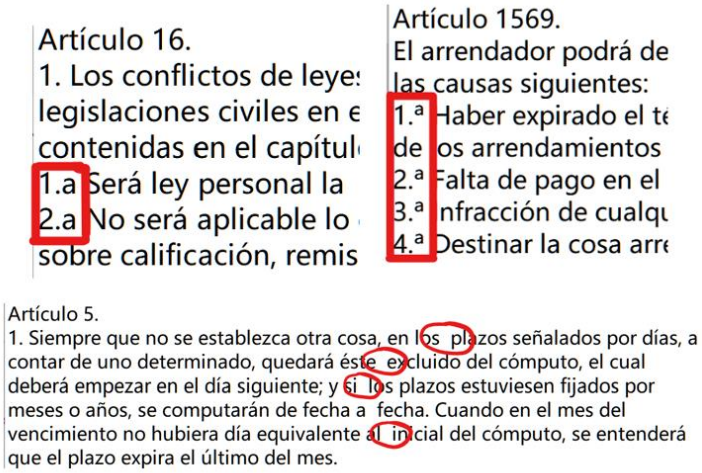

Figure 3.4 Formatting issues of the raw text.

Finally, we used Python's basic functions and objects to split the cleaned text into volumes and titles, then stored them in files and folders of the same names and structures as Figure 3.3 shows.

3.2.3. Spanish Constitution

As the Spanish Constitution is not as extensive as the Spanish Civil Code, we copied the text from the webpages and pasted them separately in "constitution_es.txt" and "constitution_zh.txt".

3.2.4. China Law Info

We downloaded a handful of files from the China Law Info database and divided them according to their section and applicable geopolitical region. We also included the citation code of China Law Info for these files in Annex I: List of Files from China Law Info.

Division	**Number of files**
Sino-foreign Treaties (Spain)	49
Sino-foreign Treaties (Europe)	57
Foreign Regulations (Europe)	11

Table 1 Statistics of downloaded files from China Law Info.

Obtained the available translation part, we started to find the possible original text. For the Sino-foreign Treaties (Spain) division, we used Google Translate to get an approximate back translation of the titles, then searched for it in Google. Though not accurate, the search results were quite promising. If not found, we turned to search directly in the *Boletín Oficial del Estado* leveraging potential critical information in the Chinese translation such as *boletín* number, publication date, and agreement signing date.

For the two remaining European parts, we approached them in the same way. However, rather than searching in Spanish, we did it in English as it usually has more exposure on the Internet. After we found the English version on EUR-Lex, we switched to the Spanish version. The well-structured nature of legal documents such as listing numbers and letters helped to identify quickly if the resource found was the source text.

Note that we did not use Joint Statement, Joint Communique, and Memorandum of Understanding. Firstly, EUR-lex does not include them. Secondly, we did not consider them strictly legal text in terms of their content and language style.

The table below shows the final result. The found and used Spanish source text can be found in the GitHub repository as well. Citation codes of these files can be found in Annex II: List of Found and Used Files from China Law Info.

Division	**Number of files**
Sino-foreign Treaties (Spain)	25
Sino-foreign Treaties (Europe)	13
Foreign Regulations (Europe)	9

Table 2 Statistics of found and used legal documents from China Law Info.

3.2.5. Sentence alignment

Sentence alignment was done manually in a CAT Tool instead of using automatic methods. Although these algorithms return good results and save time (Li et al., 2010), the time and energy to learn and apply them would have exceeded the manual way in our case as the anticipated corpus had a limited volume. Besides, manual sentence alignment could guarantee the quality, which would be vital if the amount was not advantageous.

As the Spanish and the Chinese texts were collected separately from different websites or resources, there was little HTML formatting or structural information to be leveraged for CAT tools or termbases. Thus, the initial alignment result offered by the CAT Tool was poor. However, proper configurations of the CAT Tool could accelerate this process.

We used SDL Trados Studio 2019 Professional (version 15.0.0.29074) to align documents. Three translation memories were created independently for the Civil Code, the Constitution, and the China Law Info platform documents. Each had its characteristics in terms of paragraphs, so that each requires different segmentation settings to fasten the alignment. These settings can be configured by: right-click a translation memory -> Settings -> Language Resources.

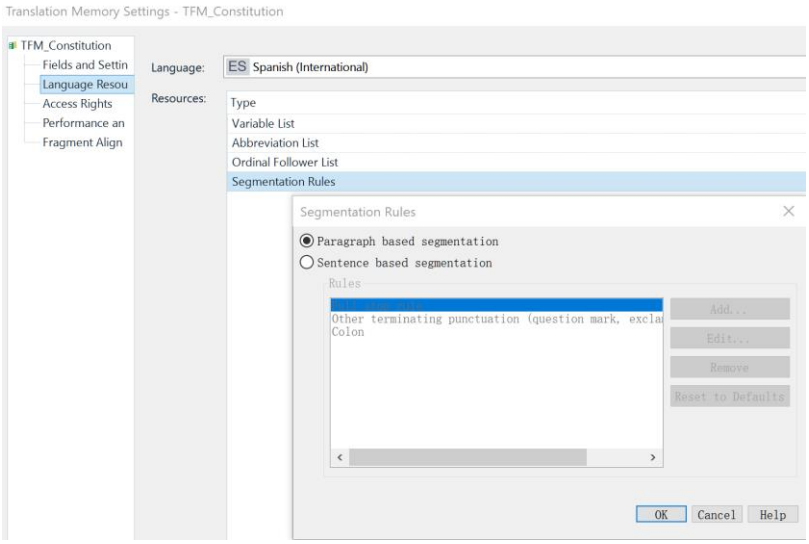

Figure 3.5 Segmentation configurations in SDL Trados Studio 2019.

As Figure 3.6 shows, chapter numbers and article numbers were separated from its content for the Spanish Civil Code. However, section number and content were separated for the Chinese text while situated in the same line in Spanish. Under such circumstances, we used the default sentence segmentation settings of translation memory in Trados, only adding one extra segmentation rule for the Spanish text to separate the content after patterns of (^Sección \d+\.ª). Remember to change to Advanced View in the rule editing window in order to use regular expressions.

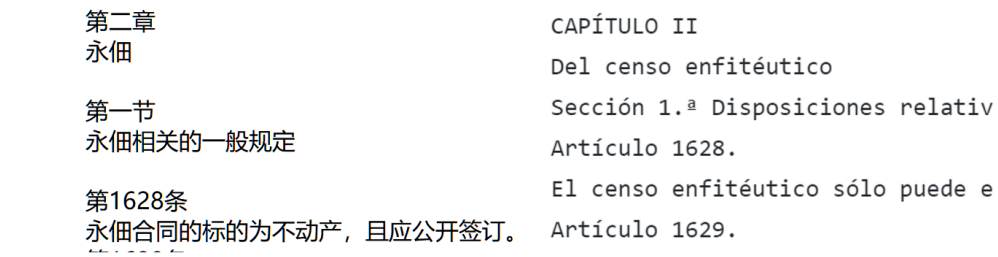

Figure 3.6 Characteristics of the text structure of the Spanish Civil Code documents.

Similarly, we added a sentence-based segmentation rule to separate the content of a line after a pattern as (^第\w+款) for Chinese; and after a pattern as (^[a-z]\)) or (^\d+\.(º|ª)) for Spanish.

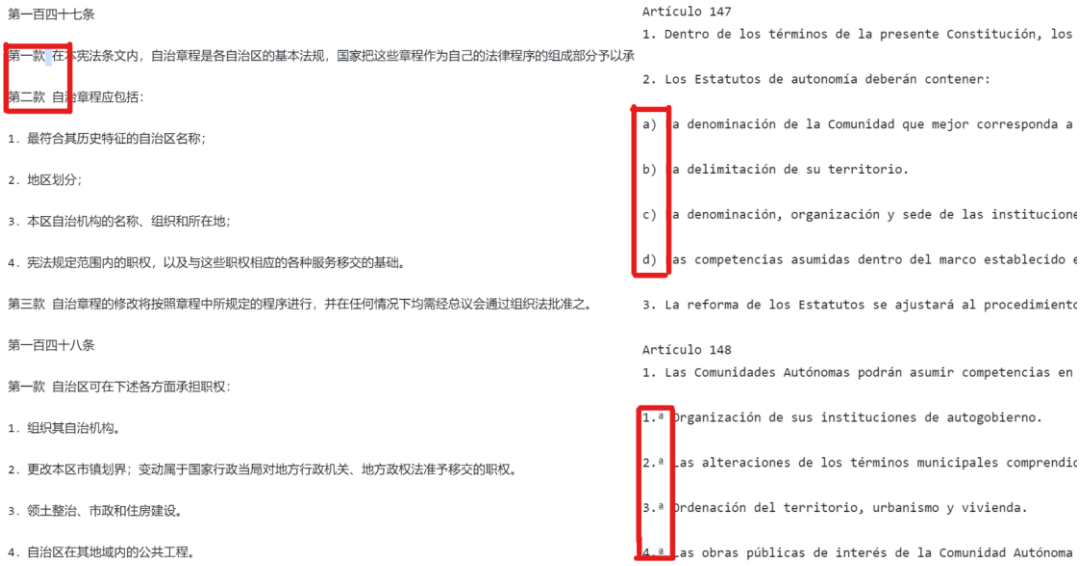

Figure 3.7 Characteristics of the text structure of the Spanish Constitution documents.

Things became even trickier for the documents related to the China Law Info database. These files hardly present shared characteristics in text structure and thus requires individual adjustments directly on the contents of a file pair instead of constantly twisting translation memory's segmentation rules. We adjusted them with the help of regex and Notepad++, including deleting extra whitespace, separating or uniting chapter/section/article numbers and their title or content, and replacing strangely formatted numbers and letters to avoid any negative influence on model training.

Figure 3.8 Example of strangely formatted number and character.

Nevertheless, we did find various segmentation rules while aligning these files in Trados Studio. For example, it was helpful to avoid segmentation after Roman numerals, used to list items and referenced in the middle of a sentence. Also, a pattern as (\d+ (bis|ter|quater)\.) should not be segmented either because they are a numbering system in the legal domain.

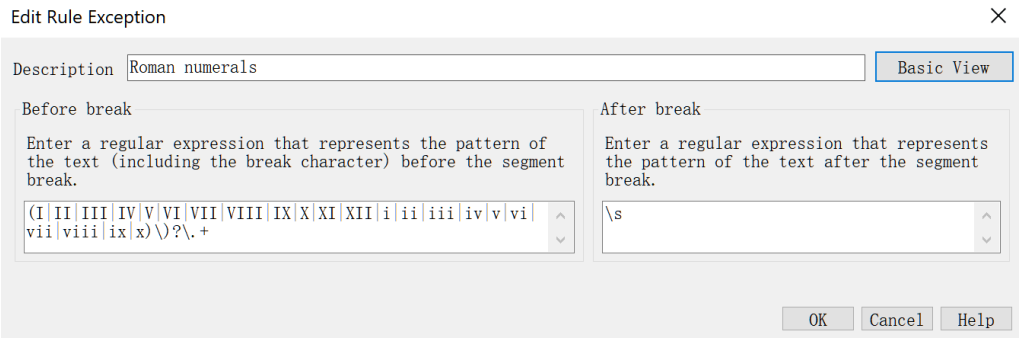

Figure 3.9 Configurations to avoid segmentation after Roman numerals.

Those adjustments to the original text or segmentation rules served the single objective of facilitating the alignment process. If a pair of files have similar structures, then SDL Trados Studio can essentially align them line by line, sentence by sentence, with high output quality. Our experience did confirm this assumption.

Besides, we developed a technique to accelerate the manual correction process. Firstly, we disconnected all automatically generated links between segments, then we manually aligned article numbers. After that, we clicked the Realign button in the Trados Studio alignment panel. One could find that forehand aligned segments serve as anchors to ensure that alignment errors produced do not influence following segments. Using this technique, we only needed to check for non 1:1 alignment after realigning, which usually implied errors in

our experience. The following figures demonstrate how correctly aligned segments can enhance automatic alignment quality.

Figure 3.10 We manually aligned segment pairs of 1082-1065 and 1085-1068. Note how alignment is incorrect for following segments, even for segment pairs of numbers.

Figure 3.11 This is the automatic realignment result. Note how numbers and contents are correctly aligned.

This technique does not necessarily require aligning article numbers. The core idea is to confirm a pair of segment alignment with an interval of segments to provide SDL Trados Studio with correct anchors. However, do be aware that this technique may only be effective if the source and target text have similar paragraph structures.

After correcting various errors of the automatic realignment result, we confirmed all links and imported the translation units to the translation memory as plain text. We left existent units unchanged, ensuring that we would not have too many duplicates or identical segments with only different placeables, for example, *Artículo 29* and *Artículo 100*. They would have little value to fine-tune our NMT model.

However, there was a disadvantage in using SDL Trados Studio to align documents. We could not select more than three segments at once of one side, meaning we needed to cut

extra text and paste them into other segments if one or two segments of one side corresponded to four or more segments on the other side.

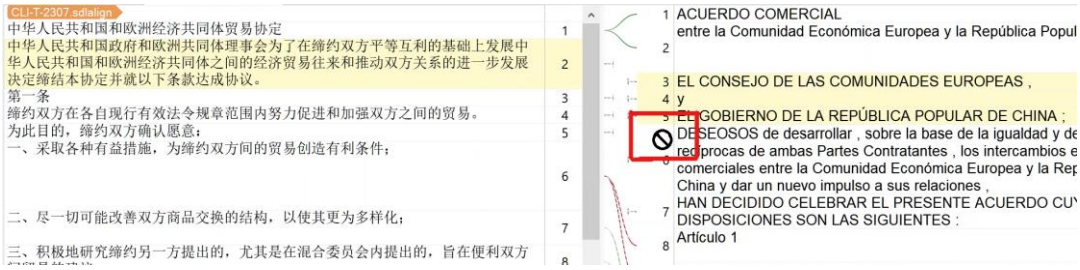

Figure 3.12 SDL Trados Studio 2019 does not support more than three segments for one side.

During this process, we also found that some files downloaded from the China Law Info platform were referring to the same law with the same content, only that they were different items in the platform. For example, *CLI-T-9111* was the same as *CLI-T-1072*; both were agreements related to preventing double taxation and tax avoidance and evasion on personal income.

Some files were all capitalized or with no accent mark; it would be noise because the tokenizer would treat them differently. We did not use these files either.

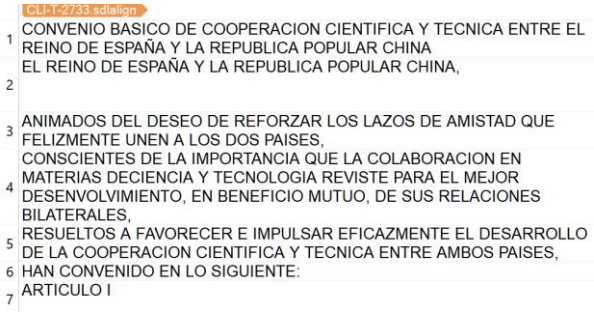

Figure 3.13 Contents fully capitalized and without accent marks.

We finally obtained 4047 segment pairs for the Spanish Civil Code, 706 for the Spanish Constitution, and 5372 for documents related to China Law Info. Then these three translation memories were exported as TMX and re-imported into a single translation memory with 9981 segment pairs in total.

We deleted various duplicates using Trados Studio's native "Search in potential duplicates only" function on this single translation memory. Then, we created a filter to export and

batch delete segments of multiple sentences. Exported segments were further segmented into single sentences and reimported into the original translation memory.

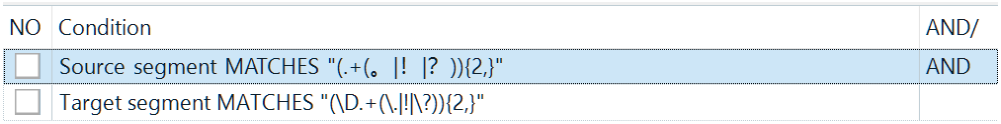

Figure 3.14 Filter conditions with regex to detect segments of multiple sentences.

Note that we considered if we actually should separate these sentences. For example, there was a source segment of "El mandato es general o especial. El primero comprende todos los negocios del mandante. El segundo, uno o más negocios determinados." with target segment of "委托有一般委托和特殊委托之分。所谓一般委托是指受托所有事务，特殊委托是指受托一项或多项事务。". The fact that *el primero* and *el segundo* are translated as '*mandato general*' and '*mandato especial*' instead of 'the first' and 'the second' in Chinese determines that the machine would need the first sentence as context to interpret the following sentences correctly, thus impedes that we separate the segment. As there are not many multi-sentence segments, we could manually decide if we should divide them into single sentences. Afterwards, we realized that performing this operation on a corpus level would be ideal when aligning segments, but still, we continued with the current corpus due to time restrictions.

We exported 44 segments of multiple sentences and then re-imported 78 segments. The final number of segment pairs in the translation memory is 9972.

3.3. Fine-tune with *transformers*

We ran codes of Hugging Face's Python libraries on Google Colab to fine-tune an NMT model from Spanish to Simplified Chinese developed by the University of Helsinki. All processing codes were uploaded to the GitHub repository. Fine-tuned model was pushed to Hugging Face Hub for open-source usage as well.

3.3.1. Fine-tuning

Fine-tuning is also a training processing for neural networks. However, different from training from scratch, it is a method to leverage an already trained model that has learned

typically on a massive amount of data for a similar task. An advantage of fine-tuning is that one can simply use a consolidated neural network without the need to construct a new one.

Fine-tuning usually freezes many initial layers of a deep neural network, only permitting the weight to be updated for some final layers (*Transfer Learning and Fine-Tuning*, 2021). However, developers of Hugging Face do argue that it is better to unfreeze the whole model for the Transformer architecture (*Fine-Tuning a Pretrained Model*, n.d.).

In whichever way, studies have shown that fine-tuning is a reasonable strategy to customize a model. For example, Jawahar *et al.* confirm that the BERT model well-captured phrasal syntax in lower layers, syntactic features in middle layers, and that "composes a hierarchy of linguistic signals ranging from surface to semantic features" (Jawahar et al., 2019). These learned weights can easily be conserved or initiated for fine-tuning, enabling us to efficiently customize a model, achieve good performance with limited data, and save time and energy.

We used Hugging Face to access pre-trained models and Google Colaboratory to conduct the fine-tuning. The original model was the only Spanish-Chinese translation model[19] found on Hugging Face Hub, developed by the University of Helsinki for its tatoeba translation challenge (Tiedemann, 2020). It was a transformer-based model trained with PyTorch and Marian frameworks.

We did not change the model configuration such as its architecture, activation function, dropout percentage, or tokenizer, which segments sentences into tokens and relates each token with an ID. Besides, we unfroze the whole model, meaning all the weights could be adjusted along each training step.

3.3.2. Hugging Face

As a company founded in 2016, Hugging Face can be viewed from different perspectives. It is a community maintained by machine learning, particularly natural language processing professionals and amateurs. It is the developer of several open-source Python libraries based on transformer neural network algorithms and optimized for easy use (Wolf et al., 2020). It

[19] https://huggingface.co/Helsinki-NLP/opus-tatoeba-es-zh

is the philosophy of facilitating access to state-of-the-art NLP models for the general public, such as BERT and GPT-2.

Although its *transformers* Python library was firstly deployed only in September 2019, it quickly transformed into a popular project on GitHub. At the time of the day, it already has collaborations with Google, Microsoft, Facebook, Amazon and many other industry leaders in machine learning and NLP.

Its most crucial library worth mentioning is the *transformers*. It gathers various cutting-edge pre-trained models to perform tasks through several code lines or fine-tune these models on their datasets. It saves an enormous amount of energy and time in collecting more extensive datasets and training from scratch. Of course, training from scratch is also possible with *transformers*.

The *transformers* library is easy to use as well. Users mainly deal with five Python classes: configurator, model, tokenizer, trainer, and pipeline.

The configurator holds information on neural networks' architecture, activation functions, dropout settings, etc. Then the model can take the configurator as a parameter and construct a neural network. Also, it saves weights information obtained from training so that it can use them for inferences. The tokenizer mainly processes textual data into numerical data to feed into the model or the trainer. As the name suggests, the trainer is used to hold together the model, tokenizer, datasets, and training arguments to train the model.

The pipeline provides an integrated manner to use a model. Usually, there are three steps to use a translation model: tokenizer encodes the string into numerical data, neural network transforms the data into another numerical data, and tokenizer decodes the numerical translation into text. With a pipeline object, one can directly pass in strings to yield translations. See the example below.

```
import transformers
model = transformers.MarianMTModel.from_pretrained('Helsinki-NLP/opus-mt-en-es')
tokenizer = transformers.MarianTokenizer.from_pretrained('Helsinki-NLP/opus-mt-en-es')
pipeline = transformers.pipeline(model=model,tokenizer=tokenizer,task='translation_en_to_es')
```

```
pipeline('Hugging Face is an active community for natural language processing.')
```

```
[{'translation_text': 'Hugging Face es una comunidad activa para el procesamiento del lenguaje natural.'}]
```

Figure 3.15 Example of how to create and use a pipeline for a translation task.

Besides the *transformers*, Hugging Face has a *datasets* library where various language data can be downloaded into structured ready to use datasets with few codes. For more information, visit the libraries' documentations on Hugging Face's website (available at: https://huggingface.co/).

3.3.3. Google Colaboratory

Google Colaboratory, or Google Colab, is a free web application for writing and executing Python 3 codes, specially designed for machine learning and data science. It requires an Internet connection and a web browser. According to its official website, most major browsers such as Chrome, Firefox and Safari are supported. Microsoft Edge also works well with it based on the personal experience of the author.

An advantage of Google Colab is that it provides users with free GPU/TPU access for heavy computation tasks. GPU type is randomly assigned among choices between Nvidia K80s, T4s, and P100s. TPU stands for tensor processing unit, which is hardware developed by Google for particular usage in tensor computations, the origin of neural network's commanding requirement of computing power. Each user generally can connect to a GPU or a TPU runtime for 12 hours on end. For faster GPU/TPU or longer runtime, the user needs to subscribe to the Colab Pro service.

Google Colab is developed upon Jupyter Notebook, an interactive application for Python coding that runs on a local machine. Thus, the file extension of a Colab notebook is also IPYNB, the open-source Jupyter notebook format. Colab notebooks are created and automatically saved on Google Drive, meaning they can be easily shared and downloaded. However, it also suggests that local files and data must be uploaded to Google Drive for Colab notebooks.

As a free and public tool, Google Colab has certain limitations on the resources. Apart from GPU/TPU speed and runtime striction mentioned before, the RAM and disk memory are also constrained and may fluctuate dynamically depending on the server load. Nevertheless, Google Colab is sufficient to general tasks out of research or educational purposes, not to mention the free access to GPU/TPU.

3.3.4. From translation memory to acceptable format

Hugging Face accepts CSV, JSON, TXT files, and pandas[20] data frames from local storage to load training samples into the runtime. In this case, we chose to import our aligned sentences from a single CSV file.

Firstly, we exported the final translation memory as a TMX file. Then, we processed the TMX file in Google Colaboratory with two built-in modules of Python, *csv* and *xml*. See the complete processing code below.

```
import xml.etree.ElementTree as et
tree = et.parse('All.tmx')
```

```
root = tree.getroot()[1]
```

```
root
```

```
<Element 'body' at 0x0000022651D68950>
```

```
import csv
```

```
with open('All.csv','w+',newline='',encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerow(['es','zh'])
    np = {'xml':'http://www.w3.org/XML/1998/namespace'}
    for tu in root:
        es = tu.find('./tuv[@xml:lang="es-x-int-SDL"]/seg',namespaces=np)
        zh = tu.find('./tuv[@xml:lang="zh-CN"]/seg',namespaces=np)
        writer.writerow([es.text,zh.text])
file.close()
```

Figure 3.16 Example Python code to extract aligned sentences from a TMX file exported from an SDL Trados Studio translation memory into a CSV file.

TMX files are essentially XML files. We used the *xml* module to first parse the whole TMX document into a nested tree object, which would have the same structure as the original document.

[20] A Python library for data manipulation and analysis.

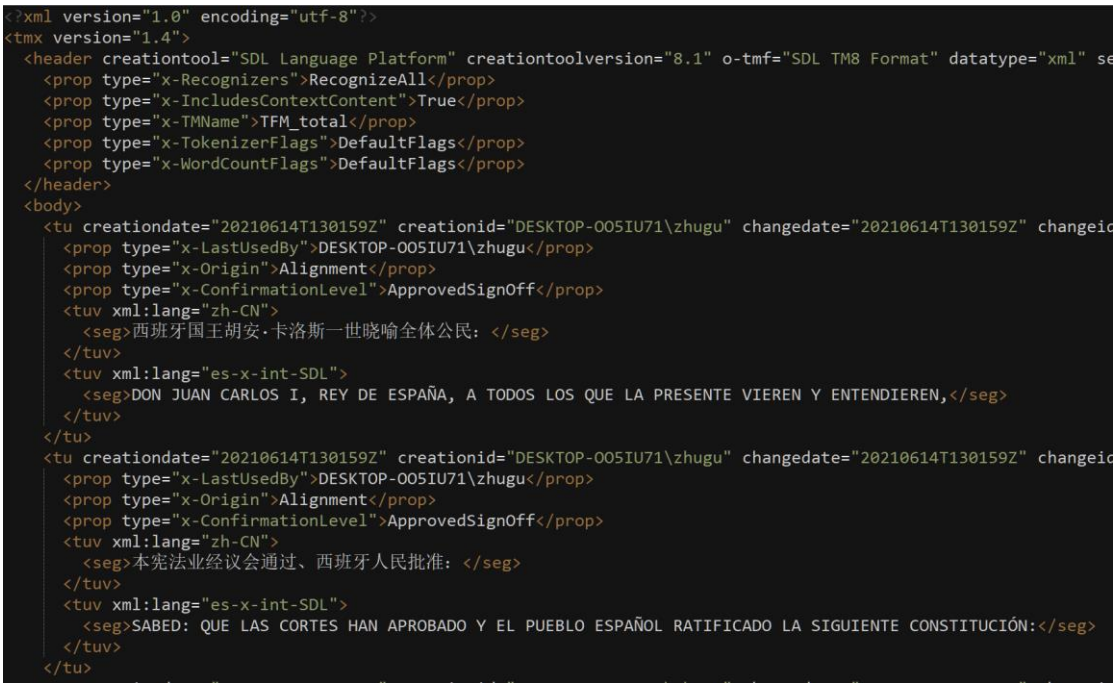

Figure 3.17 Internal structure of the TMX file.

Each sentence pair was stored within <tu></tu> tags. This tag was the first-level child element of the <body></body> tag, which we already set as the root. It was the reason why we used a simple loop algorithm to extract sentence pairs. We then used XPath syntax to find the Chinese segment and the Spanish segment elements for each pair. Finally, we extracted the content from these elements and wrote them into the CSV file. Note that we used namespaces[21] in XPath queries, which required defining it in a dictionary object beforehand. Also, we firstly wrote a row indicating the columns' names before writing segments. This first row of column names was necessary for correctly importing data into Hugging Face.

[21] Used for providing uniquely named elements and attributes in an XML document, essentially the xml:lang= "zh-CN" and the xml:lang= "es-x-int-SDL" attributes in our TMX file.

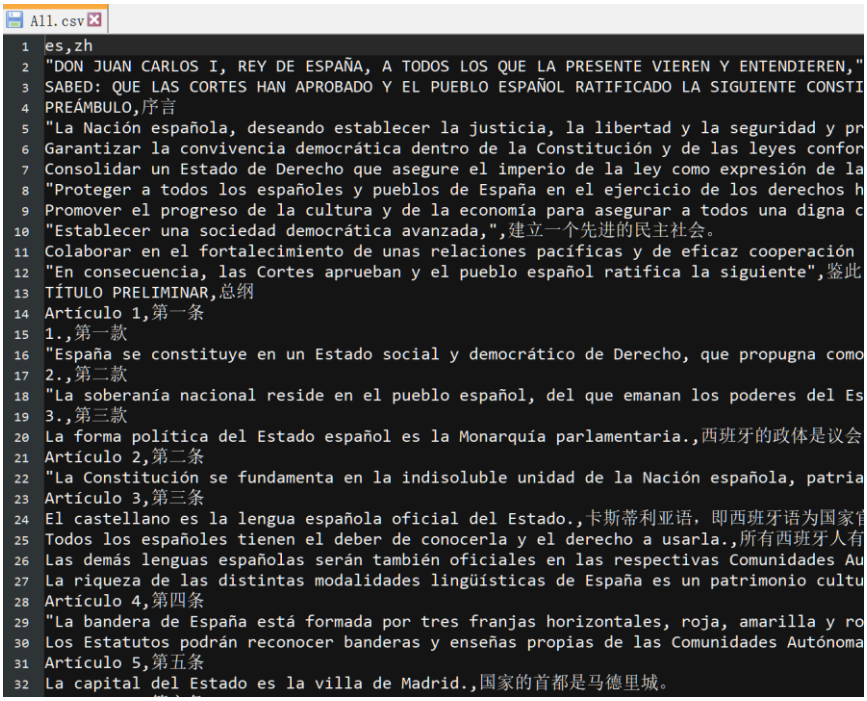

Figure 3.18 CSV file with extracted sentence pairs from the TMX file.

3.3.5. Dependencies

Even though Google Colaboratory came with a ready-to-use Python environment, it did need some runtime adjustment for our experiment.

First, we updated the *pip*, a package management system for Python, because it seemed necessary for future tokenizers to work correctly. Then, *PyYAML* also required an update. If not, errors would occur in the end when trying to push the trained model to the Hugging Face hub. Finally, we installed the *transformers*, *datasets*, and *sentencepiece* libraries, the last because our chosen pre-trained model has constructed its tokenizer on it.

3.3.6. Quickly instantiate necessary objects

The Helsinki-NLP/opus-tatoeba-es-zh model is a MarianMT model. According to the documentation of *transformers*, we would need at least a MarianMTModel instance, a MarianTokenizer instance, a DataCollatorForSeq2Seq instance, training and evaluation datasets and a Seq2SeqTrainingArguments instance to create a Seq2SeqTrainer instance so that we could use the last one to start training. Besides, a MarianConfig instance could be created to twist the model architecture and configuration.

Though it seemed complicated, it was effortless to instantiate some of those Python classes with the help of the universal method of *from_pretrained()* in the *transformers* library. As shown in the figure below, one can quickly create necessary objects with the identifier string that appears on the Hugging Face. The user only needs to figure out which Python classes they are. Note that we changed the default *use_fast* parameter from *True* to *False* when creating the tokenizer object. The so-called "fast" tokenizer is an implementation based on the Rust library tokenizers, which do not apply to all pre-trained models on Hugging Face. For example, MarianMT models do not support it.

```
from transformers import MarianConfig, MarianTokenizer, MarianMTModel
model_checkpoint = "Helsinki-NLP/opus-tatoeba-es-zh"
configurator = MarianConfig.from_pretrained(model_checkpoint)
model = MarianMTModel.from_pretrained(model_checkpoint)
tokenizer = MarianTokenizer.from_pretrained(model_checkpoint,use_fast=False)
```

Figure 3.19 Straightforward methods to instantiate configurator, model, and tokenizer classes.

We could also easily get a glimpse of the details of our model. As the figure shows below, the MarianConfig class gave information such as activation function, dropout percentage of activation and attention, number of attention heads, number of layers for the encoder and the decoder, etc. The MarianMTModel class reveals more structurally how different layers are stacked up together, with notes on the vector shape it takes and outputs. This class stores the weights as well, information which it does not show when called.

```
# we can use an instance of the configurator class to get detai
configurator
```

```
MarianConfig {
  "activation_dropout": 0.0,
  "activation_function": "swish",
  "add_bias_logits": false,
  "add_final_layer_norm": false,
  "architectures": [
    "MarianMTModel"
  ],
  "attention_dropout": 0.0,
  "bad_words_ids": [
    [
      65000
    ]
  ],
  "bos_token_id": 0,
  "classifier_dropout": 0.0,
  "d_model": 512,
  "decoder_attention_heads": 8,
  "decoder_ffn_dim": 2048,
  "decoder_layerdrop": 0.0,
  "decoder_layers": 6,
  "decoder_start_token_id": 65000,
  "do_blenderbot_90_layernorm": false,
  "dropout": 0.1,
  "encoder_attention_heads": 8,
  "encoder_ffn_dim": 2048,
  "encoder_layerdrop": 0.0,
  "encoder_layers": 6,
  "eos_token_id": 0,
```

```
model
```

```
MarianMTModel(
  (model): MarianModel(
    (shared): Embedding(65001, 512, padding_idx=65000)
    (encoder): MarianEncoder(
      (embed_tokens): Embedding(65001, 512, padding_idx=65000)
      (embed_positions): MarianSinusoidalPositionalEmbedding(512, 512)
      (layers): ModuleList(
        (0): MarianEncoderLayer(
          (self_attn): MarianAttention(
            (k_proj): Linear(in_features=512, out_features=512, bias=True)
            (v_proj): Linear(in_features=512, out_features=512, bias=True)
            (q_proj): Linear(in_features=512, out_features=512, bias=True)
            (out_proj): Linear(in_features=512, out_features=512, bias=True)
          )
          (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (fc1): Linear(in_features=512, out_features=2048, bias=True)
          (fc2): Linear(in_features=2048, out_features=512, bias=True)
          (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        )
        (1): MarianEncoderLayer(
          (self_attn): MarianAttention(
            (k_proj): Linear(in_features=512, out_features=512, bias=True)
            (v_proj): Linear(in_features=512, out_features=512, bias=True)
            (q_proj): Linear(in_features=512, out_features=512, bias=True)
            (out_proj): Linear(in_features=512, out_features=512, bias=True)
          )
          (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (fc1): Linear(in_features=512, out_features=2048, bias=True)
          (fc2): Linear(in_features=2048, out_features=512, bias=True)
          (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        )
```

Figure 3.20 Information that configurator and model objects store in the transformers *library.*

3.3.7. Prepare dataset for the training

Another central stage of the training is to build a proper dataset and tokenize it. For the CSV file that we created earlier, Hugging Face's *datasets* library supports various methods to import it into a DatasetDict instance, a dictionary of datasets. We imported it as a single split, which would be the "train" split by default. Then, we used this split as a Dataset object and divided it into "train" and "test" splits where the latter had the size of 1000. This action would automatically shuffle the whole dataset first. Note how the first row we added to the CSV file when creating it in Figure 3.16 became the features in both dataset splits.

```
DS

DatasetDict({
    train: Dataset({
        features: ['es', 'zh'],
        num_rows: 8972
    })
    test: Dataset({
        features: ['es', 'zh'],
        num_rows: 1000
    })
})
```

Figure 3.21 Final DatasetDict object we obtained.

At this point, we extracted the test dataset into two TXT files separately for the feature "es" and "zh" for later usage in BLEU calculation.

Finally, we tokenized the whole dataset dictionary with a customize function credited to the translation task example code[22] published by Hugging Face. The function adds token information of both the source and target language as features to the datasets. The textual sequence is separated as tokens and then converted into numeric ids or labels for easy computation.

```
tokenized_data['train'][50]

{'attention_mask': [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
 'es': 'Serán de cargo de la sociedad de gananciales los gastos que se
originen por alguna de las siguientes causas:',
 'input_ids': [110,6886,2,1173,2,4,574,2,45720,1955,13,472,11,20,51,17713,
5481,65,25,2094,2,15,673,2262,39,0],
 'labels': [8, 965, 2435, 971, 1214, 4851, 39, 0],
 'zh': '共同财产用于以下用途：'}
```

Figure 3.22 An example of what information each row contains from a tokenized dataset.

[22] https://github.com/huggingface/notebooks/blob/master/examples/translation.ipynb

3.3.8. Training arguments and training

Now, with almost everything ready, we could configure the training hyperparameters and start the fine-tuning process. The figure below details the hyperparameters of this experiment.

```
from transformers import Seq2SeqTrainingArguments
batch_size = 8
args = Seq2SeqTrainingArguments(
    output_dir='huggingface_outputs',
    logging_dir='huggingface_logging',
    evaluation_strategy='steps',
    prediction_loss_only = True,
    logging_steps=400,
    learning_rate=2e-5,
    warmup_steps=2000,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    weight_decay=0.01,
    save_total_limit=10,
    num_train_epochs=10,
    load_best_model_at_end=True,
    predict_with_generate=True,
    fp16=True
)
```

```
earlystopping = transformers.EarlyStoppingCallback(early_stopping_patience=8)
```

Figure 3.23 Hyperparameters of this experiment.

The training arguments object can only utilize established training epoch number to decide when to stop training. If one wants to do an early stopping before overfitting happens, a particular callback object must be created and later passed into the trainer together with other objects. For our experiment, we set the early stopping patience to 8, meaning that the training would automatically stop if there were no lower validation loss after eight validations.

Note that before we created the training argument object, we connected to the GPU runtime in Google Colaboratory and changed the working directory to Google Drive. While connecting to a GPU runtime, we are developing in a remote server, and all files generated would automatically be saved on the disk of that server. If we unfortunately disconnected from the server, the files could be lost when we had not changed the working directory to a place always accessible and saved files there.

After instantiating the data collator, we could finally create the trainer object with the model, tokenizer, training arguments, processed datasets, and callbacks. Also, the trainer uses the AdamW optimizer by default, and we did not change that.

```
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
```

```
trainer = Seq2SeqTrainer(
    model,
    args,
    train_dataset=tokenized_data['train'],
    eval_dataset=tokenized_data['test'],
    data_collator=data_collator,
    tokenizer=tokenizer,
    callbacks=[earlystopping]
)
```

```
trainer.train()
```

```
[ 1287/11220 03:02 < 23:31, 7.04 it/s, Epoch 1.15/10]
```

Figure 3.24 Creation of the data collator and the trainer.

The trainer automatically suggests a total number of 11220 steps. We could also manually calculate it. Divide the number of training examples by batch size yields the steps needed for one epoch. We have 8972 pairs of sentences as training examples. If we use a batch size of 8, meaning every step will process simultaneously eight examples, then each epoch requires 1121.5 steps, for which we need to round up the decimals, thus getting 1122 steps. Multiply it by ten epochs, and then we obtain the same result as the code returns: 11220 training steps in total.

3.3.9. Upload fine-tuned model to Hugging Face Hub

Once the training was finished, we connected our Hugging Face account to the current runtime, installed and configured Git Lfs, then simply pushed the fine-tuned model to our Hugging Face account. The model can now be downloaded and used through the *transformers* by anyone, available at: https://huggingface.co/guocheng98/HelsinkiNLP-FineTuned-Legal-es-zh.

4. Result

We first analyzed the training process and then the machine translation quality. Due to time limitations, we only resorted to the BLEU metric. We used Tilde MT's interactive BLEU score evaluator[23] to quickly obtain the metric results for the original and the fine-tuned model. Fortunately, this evaluator also generated a dynamic interface for easy-analyzing the translations segment by segment, with the help of which we examined briefly how those translations of both models differ from each other.

[23] https://www.letsmt.eu/Bleu.aspx

4.1. Fine-tuning Process

The fine-tuning process early stopped at the 7.84 epoch. It lasted 20 minutes and 54 seconds with a single NVIDIA Tesla P100 GPU. Training logs of loss is showed in Figure 4.1. The red point at step 5600 is ultimately the chosen checkpoint as our final model. The graphs correspond to Figure 2.7, indicating that we correctly approached the fine-tuning process.

However, it is also notable that the evaluation loss barely changed after a steady drop towards step 2000. We calculated in Section 3.3.8 that each epoch required 1122 steps, which means that the model no longer learned many features about the training dataset after nearly two epochs. It indicates that our training dataset may not be enough if we hope to increase the model's performance in the legal domain.

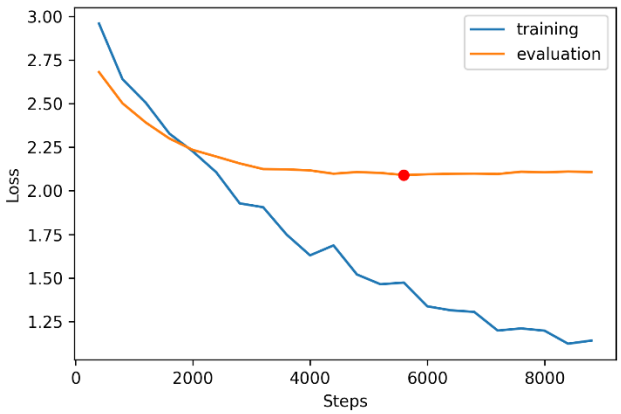

Figure 4.1 Graphs of training loss and evaluation loss.

In terms of technical experience, Hugging Face is intuitive and relatively simple to use. Especially, its documentation is clear and well-structured, which was of great help for our experiment. However, there were some issues with Google Colaboratory, particularly in dependencies, as a few built-in packages were outdated for the correct function of Hugging Face libraries.

4.2. Model Performance

With our fine-tuned model as the first machine translation engine (blue) and the original model as the second (green), Tilde's interactive BLEU score evaluator gives the following results:

BLEU:	7.29				0.39			
Precision x brevity:	7.46 x 97.72				0.39 x 100.00			
Type	1-gram	2-gram	3-gram	4-gram	1-gram	2-gram	3-gram	4-gram
Individual	5.09	9.76	10.00	6.25	1.49	0.11	0.47	0.30
Cumulative	4.98	6.89	7.74	7.29	1.49	0.40	0.42	0.39

Figure 4.2 BLEU results of the fine-tuned and the original model given by Tilde MT.

The fine-tuned model has a BLEU score of 7.29 while the original has only 0.39, with an increment of 6.90 achieved. The fine-tuned model loses a few points on brevity but is considerably better on precision. Generally, this could be interpreted as that the fine-tuned model outputs more condense and more human-like translations. Also, the new model outperforms the original one in all types of N-grams. The score of the fine-tuned model suggests productive insufficiency; however, it shows that a performance boost on domain adaptation can be attained with limited data and computation power. After all, we only used less than 10k sentence pairs as training data and trained for less than half an hour with a single GPU.

Tilde MT also displays the comparison segment by segment. Of the 1000 evaluation segments, the fine-tuned model scores better in 393 segments and worse in 30 segments. The two models have the same BLEU score for the rest of the segments.

However, the score of a single segment is not always the correct reflection of the machine translation quality, as explained in Section 2.5. For example, the original model scores higher for sentence 85, but the output of the fine-tuned model is more precise in that *potencial o real* correctly modifies *el valor comercial* instead of *la información* as the original model does. Both models propose the same translation for sentence 205, while the fine-tuned model strangely scores more than 30 points higher. Also, the fine-tuned model may still outperform the original one when the BLEU system of Tilde MT assigns them the same score. For instance, the fine-tuned model correctly translates *estado de alarma* in sentence 300; besides, its syntactic structure is more similar to the human translation.

Sentence 85	BLEU	Length ratio	Text
Source	-	-	b) el valor comercial de la información, potencial o real, en virtud de su carácter secreto;
Human	100.00	1.00	2 . 保持其为秘密可使信息具有现实的或潜在的商业价值 ;
Machine	30.93	1.67	(b) 该等信息由于其保密性质而具有的潜在或实际的商业价值 ;
Machine	50.00	1.67	(b) 可能或实际信息由于其保密性质而具有的商业价值 ;

Sentence 205	BLEU	Length ratio	Text
Source	-	-	A. Historial y evolución de la sociedad
Human	100.00	1.00	A . 公司历史及发展
Machine	84.09	1.00	A . 社会的历史和演变
Machine	50.00	1.00	A . 社会的历史和演变

Sentence 300	BLEU	Length ratio	Text
Source	-	-	Una ley orgánica regulará los estados de alarma, de excepción y de sitio, y las competencias y limitaciones correspondientes.
Human	100.00	1.00	紧急状态、特别状态和戒严状态以及有关之权力和限制， 得以一项组织法规定之。
Machine	84.09	1.67	紧急状态、特别状态和戒严状态，以及相应的权力和限制，将由组织法规定之。
Machine	84.09	0.33	一项组织法将规范警报、紧急状态和戒严状态以及相应的权力和限制。

Figure 4.3 Example translation sentences whose BLEU scores disagree with their quality.

Despite the inconsistency of the BLEU system, we can still argue that the fine-tuned model outputs better translations after analyzing many segments. A prominent improvement aspect of the fine-tuned model is how it performs better in terminology. For example, in sentence 289, *operaciones multimodales de transporte* is translated as the human translation, while the original model adds a few words to it. Likewise, in sentence 338, fine-tuned model correctly translates *documento de registro* as *注册文件* while the original model translates it as *登记文件*. The terminology is more "legal" for that the fine-tuned model translates *padres* as *双亲* instead of *父母*. Similar improvements in terminology translation can also be found in many other sentences, which we will not develop more here.

Another interesting point is that the fine-tuned model also correctly handles some textual formats. For example, it translates *Artítuclo 5* as *第五条* in sentence 287, using Chinese numbers instead of Arabic numerals as many training data indicate. Another proof is that it can now understand that *Sección 2.ª* corresponds to *第二节* rather than translating it as *第 2. a.* . Surprisingly, the fine-tuned model seems to process better when the source text is all capitalized. For example, in sentences 441 and 565, the fine-tuned model correctly translates the source texts while the original model processes *DIRECTIVA* and *VALORES* as human names.

Sentence 441	BLEU	Length ratio	Text
Source	-	-	HAN ADOPTADO LA PRESENTE DIRECTIVA:
Human	100.00	1.00	特通过本指令：
Machine	84.09	1.00	兹协议如下：
Machine	84.09	1.00	德列克特瓦总统通过：

Sentence 565	BLEU	Length ratio	Text
Source	-	-	NOTA SOBRE LOS VALORES
Human	100.00	1.00	证券说明
Machine	84.09	1.00	标的物的说明
Machine	84.09	1.00	关于瓦奥雷斯的说明

Figure 4.4 Examples of how the fine-tuned model handles better all capitalized source texts.

However, the results of the fine-tuned model are not robust, which is a characteristic of neural machine translation models. For example, the original model has better translations for sentences 325 and 413. There are wrong translations, such as translating *el Gobierno de la República de Corea* to *the Government of the People's Republic of China* as if the model does not understand the text at all. The fine-tuned model can also omit text. For example, in sentence 439, it does not translate *adheridas al suelo*.

Sentence 325	BLEU	Length ratio	Text
Source	-	-	A. Controles de cambio
Human	100.00	1.00	A . 外汇管制
Machine	84.09	1.00	A . 汇兑控制
Machine	50.00	1.00	A . 外汇管制

Sentence 340	BLEU	Length ratio	Text
Source	-	-	por el Gobierno de la República de Corea por el Gobierno de la Federación de Rusia
Human	100.00	1.00	大韩民国政府 俄罗斯联邦政府
Machine	30.93	0.50	中华人民共和国政府和中华人民共和国政府
Machine	30.93	0.50	俄罗斯联邦政府给大韩民国政府

Sentence 413	BLEU	Length ratio	Text
Source	-	-	(27) Las agencias de calificación crediticia deben evitar las situaciones de conflicto de intereses y gestionar tales conflictos adecuadamente cuando resulten ineludibles, al objeto de garantizar su independencia.
Human	100.00	1.00	（27） 信用评级机构应避免发生利益冲突， 并在无法避免冲突时对冲突进行适当管理以确保自身独立性。
Machine	84.09	1.00	(27) 为确保信用评级机构的独立性，信用评级机构应避免并妥善管理不可避免的利益冲突情况。
Machine	50.00	1.33	(27) 信用评级机构必须避免利益冲突情况，在不可避免的情况下适当管理利益冲突情况，以确保其独立性。

Sentence 439	BLEU	Length ratio	Text
Source	-	-	1.° Las tierras, edificios, caminos y construcciones de todo género adheridas al suelo.
Human	100.00	1.00	1 土地、建筑物、道路和各种与地面紧密联系的结构。
Machine	84.09	1.00	1 土地、建筑物、公路和各种建筑。
Machine	50.00	2.00	1. 土地、建筑物、道路和建筑，所有性别均附在地上。

Figure 4.5 Examples where the fine-tuned model performs worse.

4.3. Limitations

In terms of the fine-tuning process, a significant shortcoming is that the training dataset was not of enough volume. The model could have learned more about legal texts, which was also reflected in the stagnation of the evaluation loss after 2000 training steps. Even the alignment quality was good due to the manual method; however, the parallel corpus was not homogenous in that there were formatting and translational inconsistencies. It might cause noise to the training process as well. Besides, we barely changed the neural network's structure of the original model, nor hyperparameters such as dropout percentage and activation function.

On the other hand, the automatic translation evaluation metric of BLEU was far from satisfying. The metric generally reflected the enhancement of translations on the whole evaluation dataset. However, it did not always agree with our human judgment on a sentence level, which is warned in the original paper to propose BLEU (Papineni et al., 2002). Moreover, we only examined part of the test dataset on a superficial level due to the time and cost limitations of this master thesis. It should have been better to resort to human post-editing on the outputs of both models to gain more scientific and systematic insights into the comparison of adequacy, fluency, and productivity.

In the future, we may explore how we can enhance the model performance by twisting the neural network. For example, freeze the first layers after the embedding layer, deepen the neural network, and use different optimizers. Also, we could use data augmentation and other data cleaning techniques to obtain a more extensive and qualified training dataset to fine-tune the original model better. Models' performance evaluation process can be improved as well. For example, contact human translators and reviewers to post-edit and evaluate the raw MT outputs.

5. Conclusion

We explained in detail how neural networks work, especially the Transformer architecture as the state-of-the-art. Applied some notions we have seen during the theoretical framework chapter, we also successfully fine-tuned a neural machine translation model from Spanish to Simplified Chinese, seeking to adapt its domain from general to legal with our parallel corpus. Although the bitext volume was not abundant, it was still considerably efficient

compared to training a model from scratch, which requires vast amounts of data and energy. The fine-tuned model performed better in various aspects, with particular improvement in terminology, but maintained the internal drawback of lack of robustness as an NMT model, with obvious or even ridiculous errors that human translators would never commit.

Although Hugging Face lowers the threshold of developing a neural machine translation system, one still needs to understand the essence of neural networks and grasp the know-how of different training situations, apart from basic programming knowledge, in order to correctly twist the hyperparameters or adjust the network structure for better training results. We also had to admit the efforts required for a non-technical background professional to comprehend the inside mechanism of this black box of NMT.

Moreover, as a data-driven approach to machine learning, neural machine translation requires a significant volume of training examples, which we found of at least equal importance as the training process. The data availability directly determines if the neural network is feasible. There is no extensive parallel corpus ready-to-use for Spanish and Simplified Chinese, let alone a corpus of a specific domain. However, there are many resources exploitable on the Internet such as private databases and e-books. This data scarcity also suggests that fine-tuning would be more practical for developing domain-specific NMT systems.

In conclusion, neural machine translation requires both linguistic data availability and technical know-how. Nevertheless, language industry professionals are not necessarily incapable of educating themselves of NMT. We believe it would be beneficial to understand it to hold a correct attitude towards it and know when and how to leverage it.

Bibliography

Alammar, J. (2018, June 27). *The Illustrated Transformer*. http://jalammar.github.io/illustrated-transformer/

Allibhai, E. (2018, October 3). *Holdout vs. Cross-validation in Machine Learning*. Medium. https://medium.com/@eijaz/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer Normalization. *ArXiv:1607.06450 [Cs, Stat]*. http://arxiv.org/abs/1607.06450

Baker, M., & Saldanha, G. (2019). *Routledge Encyclopedia of Translation Studies*. Routledge. https://doi.org/10.4324/9781315678627

Banerjee, S., & Lavie, A. (2005). METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 65–72. https://www.aclweb.org/anthology/W05-0909

Bernard Vauquois. (2003). Automatic translation. A survey of different approaches. In *Readings in machine translation* (pp. 333–337).

Callison-Burch, C., Osborne, M., & Koehn, P. (2006, April). Re-evaluating the Role of Bleu in Machine Translation Research. *11th Conference of the European Chapter of the Association for Computational Linguistics*. EACL 2006, Trento, Italy. https://www.aclweb.org/anthology/E06-1032

Chéragui, M. A. (2012). Theoretical Overview of Machine translation. *ICWIT*, 160–169.

Costa-Jussà, M. R., Aldón, D., & Fonollosa, J. A. R. (2017). Chinese–Spanish neural machine translation enhanced with character and word bitmap fonts. *Machine Translation*, *31*(1), 35–47. https://doi.org/10.1007/s10590-017-9196-0

Fine-tuning a pretrained model. (n.d.). Transformers 4.7.0 Documentation. https://huggingface.co/transformers/training.html

Forcada, Mikel L. "Making Sense of Neural Machine Translation." *Translation Spaces*, vol. 6, no. 2, 2017, pp. 291–309. *DOI.org (Crossref)*, doi:10.1075/ts.6.2.06for.

Francois Chaubard, Michael Fang, Guillaume Genthial, Rohit Mundra, & Richard Socher. (2019). Word Vectors I: Introduction, SVD and Word2Vec. *Class Note for CS224n: Natural Language Processing with Deep Learning, Department of Computer Science, Stanford University, Palo Alto, CA, USA, Winter 2019*. https://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes01-wordvecs1.pdf

Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing*. Morgan & Claypool Publishers.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.

Guresen, E., & Kayakutlu, G. (2011). Definition of artificial neural networks with comparison to other networks. *Procedia Computer Science*, *3*, 426–433. https://doi.org/10.1016/j.procs.2010.12.071

Huang, Z., Zweig, G., Levit, M., Dumoulin, B., Oguz, B., & Chang, S. (2013). Accelerating recurrent neural network training via two stage classes and parallelization. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, 326–331. https://doi.org/10.1109/ASRU.2013.6707751

Jawahar, G., Sagot, B., & Seddah, D. (2019). What Does BERT Learn about the Structure of Language? *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3651–3657. https://doi.org/10.18653/v1/P19-1356

Kaplan, R. M. (2005). A method for tokenizing text. In *Inquiries into words, constraints and contexts* (pp. 55–64).

Koehn, P. (2020). *Neural Machine Translation*. Cambridge University Press. https://doi.org/10.1017/9781108608480

Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling* (1st ed. 2013, Corr. 2nd printing 2018 edition). Springer.

Li, P., Sun, M., & Xue, P. (2010). Fast-Champollion: A Fast and Robust Sentence Alignment Algorithm. *Coling 2010: Posters*, 710–718. https://www.aclweb.org/anthology/C10-2081

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. https://doi.org/10.1017/CBO9780511809071

Mathur, N., Baldwin, T., & Cohn, T. (2020). Tangled up in BLEU: Reevaluating the Evaluation of Automatic Machine Translation Evaluation Metrics. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4984–4997. https://doi.org/10.18653/v1/2020.acl-main.448

Mehryar Mohri, Afshin Rostamizadeh, & Ameet Talwalkar. (2018). *Foundations of Machine Learning* (2nd ed.). MIT Press. https://cs.nyu.edu/~mohri/mlbook/

Mikolov, T., Yih, W., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 746–751. https://www.aclweb.org/anthology/N13-1090

Ouyang, P., Yin, S., & Wei, S. (2017). A fast and power efficient architecture to parallelize LSTM based RNN for cognitive intelligence applications. *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 1–6. https://doi.org/10.1145/3061639.3062187

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu: A Method for Automatic Evaluation of Machine Translation. *Proceedings of the 40th Annual Meeting of the*

Association for Computational Linguistics, 311–318. https://doi.org/10.3115/1073083.1073135

Pérez-Ortiz, J. A., Forcada, M. L., & Sánchez-Martínez, F. (2021). How neural machine translation works. In *Machine Translation for Multilingual Citizens*. https://drive.google.com/file/d/1qgBt7T9nc02A9l5okC9b_XqBw32k8Ivd/view

Phi, M. (2020, May 1). *Illustrated Guide to Transformers- Step by Step Explanation*. Medium. https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

Poibeau, T. (2017). *Machine Translation*. MIT Press.

Popel, M., & Bojar, O. (2018). Training Tips for the Transformer Model. *The Prague Bulletin of Mathematical Linguistics*, *110*(1), 43–70. https://doi.org/10.2478/pralin-2018-0002

Ruder, S. (2017). An overview of gradient descent optimization algorithms. *ArXiv:1609.04747 [Cs]*. http://arxiv.org/abs/1609.04747

Sánchez Ramos, M. del M., & Rico Pérez, C. (2020). *Traducción automática: Conceptos clave, procesos de evaluación y técnicas de posedición*. Comares. https://abacus.universidadeuropea.es/handle/11268/9377

Smilkov, D., Thorat, N., Nicholson, C., Reif, E., Viégas, F. B., & Wattenberg, M. (2016). Embedding Projector: Interactive Visualization and Interpretation of Embeddings. *ArXiv:1611.05469 [Cs, Stat]*. http://arxiv.org/abs/1611.05469

Spanish Parliament. (2013). *Xibanya Minfadian 西班牙民法典 [Spanish Civil Code] (D. Pan & Q. Ma, Trans.)* (1st ed.). China University of Political Science and Law Press.

Tiedemann, J. (2020). The Tatoeba Translation Challenge—Realistic Data Sets for Low Resource and Multilingual MT. *ArXiv:2010.06354 [Cs]*. http://arxiv.org/abs/2010.06354

Transfer learning and fine-tuning. (2021, June 17). TensorFlow. https://www.tensorflow.org/tutorials/images/transfer_learning

Turian, J. P., Shea, L., & Melamed, I. D. (2006). *Evaluation of Machine Translation and its Evaluation*. Defense Technical Information Center. https://doi.org/10.21236/ADA453509

Vasani, D. (2019, November 18). *This thing called Weight Decay*. Medium. https://towardsdatascience.com/this-thing-called-weight-decay-a7cd4bcfccab

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *ArXiv:1706.03762 [Cs]*. http://arxiv.org/abs/1706.03762

Vig, J. (2019). A Multiscale Visualization of Attention in the Transformer Model. *ArXiv:1906.05714 [Cs]*. http://arxiv.org/abs/1906.05714

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., … Rush, A. M. (2020). HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv:1910.03771 [Cs]*. http://arxiv.org/abs/1910.03771

Yadav, N., Yadav, A., & Kumar, M. (2015). History of Neural Networks. In N. Yadav, A. Yadav, & M. Kumar, *An Introduction to Neural Network Methods for Differential Equations* (pp. 13–15). Springer Netherlands. https://doi.org/10.1007/978-94-017-9816-7_2

Annex

I: List of Files from China Law Info

Sino-foreign Treaties (Spain): CLI-T-10112, CLI-T-1072, CLI-T-1164, CLI-T-1409, CLI-T-184, CLI-T-1956, CLI-T-2125, CLI-T-2285, CLI-T-2310, CLI-T-257, CLI-T-2580, CLI-T-2733, CLI-T-2734, CLI-T-2956, CLI-T-2982, CLI-T-3121, CLI-T-364, CLI-T-3773, CLI-T-3937, CLI-T-3958, CLI-T-3985, CLI-T-3986, CLI-T-4018, CLI-T-4029, CLI-T-4733, CLI-T-5085, CLI-T-5225, CLI-T-5226, CLI-T-5229, CLI-T-5480, CLI-T-5523, CLI-T-6002, CLI-T-6146, CLI-T-6156, CLI-T-6605, CLI-T-6897, CLI-T-6934, CLI-T-7055, CLI-T-7382, CLI-T-746, CLI-T-7863, CLI-T-7864, CLI-T-7977, CLI-T-8337, CLI-T-9022, CLI-T-9111, CLI-T-9211, CLI-T-9212, CLI-T-9726

Sino-foreign Treaties (Europe): CLI-T-2307, CLI-T-2772, CLI-T-3030, CLI-T-3195, CLI-T-3633, CLI-T-3869, CLI-T-3886, CLI-T-3918, CLI-T-3922, CLI-T-3926, CLI-T-4025, CLI-T-4030, CLI-T-4049, CLI-T-4102, CLI-T-4243, CLI-T-4538, CLI-T-4540, CLI-T-4541, CLI-T-5161, CLI-T-5521, CLI-T-5748, CLI-T-6100, CLI-T-6109, CLI-T-6354, CLI-T-6374, CLI-T-6376, CLI-T-6479, CLI-T-6582, CLI-T-6782, CLI-T-6783, CLI-T-6808, CLI-T-6844, CLI-T-7139, CLI-T-7163, CLI-T-7200, CLI-T-7270, CLI-T-7273, CLI-T-7387, CLI-T-7388, CLI-T-7389, CLI-T-7390, CLI-T-7391, CLI-T-7392, CLI-T-7732, CLI-T-7773, CLI-T-7873, CLI-T-8005, CLI-T-8718, CLI-T-8727, CLI-T-8868, CLI-T-8869, CLI-T-9034, CLI-T-9035, CLI-T-9050, CLI-T-9079, CLI-T-9208, CLI-T-9685

Foreign Regulations (Europe): CLI-FL-1178, CLI-FL-1179, CLI-FL-1183, CLI-FL-1228, CLI-FL-1229, CLI-FL-1230, CLI-FL-1231, CLI-FL-1232, CLI-FL-1234, CLI-FL-1235, CLI-FL-1236

II: List of Found and Used Files from China Law Info

Sino-foreign Treaties (Spain): CLI-T-1072, CLI-T-1164, CLI-T-184, CLI-T-1956, CLI-T-2285, CLI-T-257, CLI-T-2580, CLI-T-2733, CLI-T-2956, CLI-T-364, CLI-T-3773, CLI-T-3958, CLI-T-3985, CLI-T-3986, CLI-T-4018, CLI-T-5085, CLI-T-5225, CLI-T-6002, CLI-T-6605, CLI-T-6897, CLI-T-7055, CLI-T-746, CLI-T-9111, CLI-T-9211, CLI-T-9212

Sino-foreign Treaties (Europe): CLI-T-2307, CLI-T-2772, CLI-T-3633, CLI-T-4030, CLI-T-4102, CLI-T-4243, CLI-T-4541, CLI-T-5748, CLI-T-6374, CLI-T-6376, CLI-T-6479, CLI-T-6808, CLI-T-9208

Foreign Regulations (Europe): CLI-FL-1178, CLI-FL-1179, CLI-FL-1228, CLI-FL-1229, CLI-FL-1230, CLI-FL-1231, CLI-FL-1232, CLI-FL-1234, CLI-FL-1235