UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Streaming neural machine translation systems from English into European languages

DEGREE FINAL WORK

Degree in Computer Engineering

Author: Guillem Calabuig Domenech

Tutor: Jorge Civera Saiz
Javier Iranzo Sánchez

Course 2021-2022

Resum

????

Paraules clau: aprenentatge automátic, traducció automática, aprenentatge profund

Resumen

????

Palabras clave: aprendizaje automático, traducción automática, aprendizaje profundo

Abstract

nmt

Key words: machine learning, machine translation, deep learning, natural language processing

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

????? ????????????? ????????????? ????????????? ????????????? ?????????????

1.1 Motivation

????? ????????????? ????????????? ????????????? ????????????? ?????????????

1.2 Goals

????? ????????????? ????????????? ????????????? ????????????? ?????????????

1.3 Machine Learning

Artificial Inteligence is built upon the grounds of computers running instructions, as any other kind of instructions that they would run, but these in appearence resembling actions that are atributed to intelligence and in many ocassions mimicking tasks done by humans.

The present work belongs to the branch of Articial Intelligence called Machine Learnig. In ~~difference~~ contrast to other branches, the intelligent behaviour of a system is achieved based mainly on the data that we make available to the system.

We could see the objective of a Machine Learning system as the task of offering correct answers to a certain set of questions. Let these questions be of any kind, whether a patient suffers from cancer or not, should the car turn left, right or stop, which is the next movie that we should recommend a user to watch, etc. In the task that ensues us our systems will be trying to give proper answers to the question of what is a good translation of a sentence into another language.

A system gives initially answers that are very distant to what we consider ~~true~~ correct, and as it processes the ~~information~~ data that we have provided to it, the performance is increased. Thus we say that the system has *learnt* and learnt from data.

In its basis a ML system is a computer program that materializes (in a big physical arena of transistors) a mathematical model, often a function, to which we give as argument our question and computes the answer depending on a set of *weights*, operations and structure. These weights vary their values through an optimization proccess that depends on data, to which we refer as *training*. Learning is thus the result of the system

having found a set of values for the weights that make the model offer better answers to the problem presented.

We can recall a referential definition of Machine Learning by Tom Mitchell [26]:

"A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"

A function is optimized from data E to answer questions T these being more or less correct P. A lot of engineering work, scientific research and experimentation is performed in order to discover better weight-keeping structures, model configurations and optimization processes that allow to come up with better weights (learning) that achieve better performance (more useful answers) to a specific task (question) from experience (data).

1.3.1. Supervised learning and Classification

Many problems or question-answering instances can be viewed as a *classification* task, that falls into the branch of Machine Learning known as *supervised learning*. In this scenario, data or experience to learn from comes as a pair $(\mathbf{x}, \mathbf{C})$, $\mathbf{x}$ being the task that we are to solve and $\mathbf{C}$ being the solution to the task, the correct answer or *class* to which $\mathbf{x}$ should be classified to.

We keep in mind that the usefulness of a system and the ultimate goal always comes in aswering questions for which there is uncertainty, inputs $\mathbf{x}$ for which the right class $\mathbf{C}$ is not known. But the model will learn to improve the performance of this task from the knowledge that is present in well-answered question pairs, correct solutions to instances of the same kind of problems.

We are refering here to performing *inference*. A model that generalizes, being able to offer a solution to a question for which the answer has not yet seen. Output a correct classification to new samples of the task, that have not been involved during the training of the model.

The final objective is to construct a system where false answers are minimized. In other words, provide solutions to tasks T so that performance P is maximized, i.e. minimize the number of mistaken classifications.

Bayes classifier

We can define $p(error|\mathbf{x})$ as the probability of sample $\mathbf{x}$ to be classified to an incorrect class and $p(c|\mathbf{x})$ the probability of $\mathbf{x}$ to belong to class c.

$$p(error|\mathbf{x}) = 1 - p(\mathbf{C} = c|\mathbf{x}) \tag{1.1}$$

And pick the label that minimizes the error.

$$\min p(error|\mathbf{x}) = 1 - \max_{c \in C} p(c|\mathbf{x}) \tag{1.2}$$

$$\mathbf{c} = \arg\max_{c \in C} p(c|\mathbf{x}) \tag{1.3}$$

Where C is the set of possible class labels and $\mathbf{c}$ the class decided by the system. We refer to this as the minimum risk decision rule or Bayes classifier.

If the true distribution $p(c|\mathbf{x})$ was known for every $\mathbf{x}$, we would have an optimal performance, an oracle, since $\mathbf{x}$ would always be mapped to the most probable class, thus other asigments incurring in more errors.

We could understand ML as a way where we're building a model that tries to approximate better these distributions, using for such task the information that is available in the data supplied to our system. Many times the use of Bayes rule can be seen to deduce an expression that is easier to approximate by some models

$$\arg\max_{c\in C} p(c|\mathbf{x}) = \arg\max_{c\in C} \frac{p(\mathbf{x}|c)p(c)}{p(\mathbf{x})} = \arg\max_{c\in C} p(\mathbf{x}|c)p(c) \tag{1.4}$$

where $p(\mathbf{x})$ is dropped since it is a constant term that doesn't modify the argument result of the optimization.

Iris dataset example

Consider the case where we had only 2 possible outcomes $c \in \{0, 1\}$. Just 2 possible answers. Taking the **Iris dataset** as an example, a question could be, given the petal length and petal width of this flower (as vector $\mathbf{x} \in \mathbb{R}^2$), is it a virginica type flower or isn't?

We then could model this uncertainty under a Bernoulli distribution. The probability that the flower is virginica would be computed depending on the information of the petal, and a set of weights $\mathbf{w} \in \mathbb{R}^d$. These weights, would have been learnt from our data, many flowers, of which their attributes have been measured, and humanly judged their class virginica or not.

$$p(c|\mathbf{x}, \mathbf{w}) = Ber(c|\sigma(\mathbf{w}^T\mathbf{x} + b)) \tag{1.5}$$

$$p(c = 1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}} \in [0, 1] \tag{1.6}$$

We would then predict flower $\mathbf{x}$ to be viginica (c=1) if $p(c = 1|\mathbf{x}) > p(y = 0|\mathbf{x})$.

This classifier would define a *linear decision boundary* a line on the 2-d Iris flowers' space that forms 2 regions, one where flowers are of the type virginica and another region that are of other type. The boundary would be a plane instead of a line if the representation of our data was 3-d, or an hyperplane in a d-dimensional space.

Similarly, we could extend our problem to allow N classes, and estimating the set of possible outcomes with a categorical distribution. We replace the previous σ sigmoid function by the *softmax* function S.

$$p(c|\mathbf{x}, \mathbf{W}) = Cat(c|S(\mathbf{W}\mathbf{x} + \mathbf{b})) \tag{1.7}$$

This would still separate the space in regions defined by several linear decision boundaries. If every sample $\mathbf{x}$ (an iris flower in our example) can be classified without error under a linear decision boundary, data is said to be linearly separable.

Perceptron algorithm

We've seen how estimating a probability distribution can result in ripping the space into regions when it comes down to classifiying inputs to class labels (tasks to their solutions)

The Perceptron algorithm [30] was a fundamental precursor of neural networks, and it focuses directly in computing a linear decision boundary for a binary classification task. This translates in finding a good set of weights in the search space of their values, that will define the hyperplane of the boundary.

I would not include this as argument of the function

We've got the following prediction model:

$$\mathbf{c} = f(\mathbf{x}) = \mathbb{I}(\mathbf{w}^T\mathbf{x} + b > 0) = \begin{cases} 1 \text{ if } \mathbf{w}^T\mathbf{x} + b > 0, \\ 0 \text{ otherwise} \end{cases} \tag{1.8}$$

The basic idea of the algorithm is to start with random weights, iteratively predict an output for each one of our labeled data $(\mathbf{x}, \mathbf{C})$, and update the weights when the model makes a mistake.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda(\mathbf{c}_t - \mathbf{C}_t)\mathbf{x}_t \tag{1.9}$$

Where λ is the *learning rate* or step size.

If the model predicts the correct label at time t, no change is made in the weights from this sample, otherwise weights are moved in a direction so as to make correct predictions more likely. If $\mathbf{c}_t = 1$ was predicted incorrectly, we have $\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda\mathbf{x}_t$. If $\mathbf{c}_t = 0$ was predicted incorrectly, we have $\mathbf{w}_{t+1} = \mathbf{w}_t + \lambda\mathbf{x}_t$.

The Perceptron was prooven to converge when data is linearly separable.

1.3.2. Deep learning

Multilayer perceptron

You need to link this sentence with the previous section saying something like: "... problem perceptrons, presented in the previous section, cannot solve."

The **XOR Problem** from *Perceptrons* [25] was a classic example of a problem perceptron cannot solve. A basic case where data isn't linearly separable. We want to reperesent a function that computes the exclusive OR. So given $\mathbf{x} \in \{(0,0),(0,1),(1,0),(1,1)\}$, $\mathbf{c}$ must equal to 1 only if one of the elements of the input is 1. As seen in figure (xor problem), no line could be drawn, so that samples get correctly classified.

* linear boundary and 0, otherwise

defined

However, with a perceptron, we can model an OR function, and with another one, an AND function since these are linearly separable cases. We could then send our sample $\mathbf{x}$ to the OR and the AND.

This is a nice example to show how a multilayer perceptron can solve a task (XOR) that a single perceptron can't, but you need to go into details and devote one page with plots and a diagram to be understood

$$\phi(\mathbf{x})_0 = \mathbb{I}(\mathbf{w}_{AND}^T\mathbf{x} + b_{AND} > 0) \tag{1.10}$$

$$\phi(\mathbf{x})_1 = \mathbb{I}(\mathbf{w}_{OR}^T\mathbf{x} + b_{OR} > 0) \tag{1.11}$$

As a result, we would have transformed our input, to another possible intermediate state $\phi(\mathbf{x}) \in \{(0,0),(0,1),(0,1),(1,1)\}$, where the first coordinate now represents whether the AND was true, and second coordinate whether the OR was true.

$$\phi(\mathbf{x}) = \mathbb{I}(\mathbf{W}\mathbf{x} + \mathbf{b} > 0) \tag{1.12}$$

belongs

Now, only the (0, 1) pertains to class 1 (or but not and, thus xor), with which we see that the problem is now linearly separable.

$$f_{XOR}(\mathbf{x}) = \mathbb{I}(\mathbf{w}_{XOR}^T\phi(\mathbf{x}) + b_{XOR} > 0) \tag{1.13}$$

We have transformed a nonlinear separable problem, not solvable by a perceptron, to a linear separable problem, just by joining units of them with specific weight settings (different weights resembling different functions and then different information extracted from the raw inputs in each perceptron unit).

is

This power of merging multiple simple units to overcome far complex problems is the heart idea laying behind neural networks. We have seen in the previous example what is known as a multilayer perceptron. Here, a perceptron resembles what it is

Not sure whether this is a * proper collocation

A possible solution is to simplify the explanation to convey the basic idea

You would need to put some effort here and I don't know if you have the time

known in deep learning as a neuron. The difference is, that perceptron computes a non differentiable function with respect to the inputs received (see Equation 1.8) whereas in neural networks, differentiable *activation functions* are used.

It is not a good idea to start a sentence with "and"

And this is key, because in difference to the example that has been ilustrated, the goal ~~and wish~~ of a neural net structure, is ~~it~~ to learn the weights of each unit automatically. Let the model and the optimization process decide which are the weights, for building better functions that transform the input into a problem that is closer to become linearly separable in each *layer*~~, until the knot is untied at the end of the whole pass through the net~~.

The model optimization will always depend on the experience that is provided to the system. So it is the data that has been humanly *supervised* what will allow the new unseen data for which we want to predict an outcome to receive a proper answer. And the model being a means intermediary facilitating this process.

I would talk about $f(x)$ as an arbitrary complex function that can be understood as a simpler functions that are nested

Neural network

In a neural network we can have many transformations in chain

$$f(x) = f_L(f_{L-1}(\cdots(f_1(\mathbf{x}))\cdots)) \tag{1.14}$$

Remember use equations as words in a sentence

Where each layer f_l has its own parameters $\mathbf{W}_l, \mathbf{b}_l$ and a non-linear differentiable *activation* function a instead of the heaviside function $\mathbb{I}$.

$$\mathbf{h}_l = f_l(\mathbf{h}_{l-1}) = a(\mathbf{W}_l\mathbf{h}_{l-1} + \mathbf{b}_l) \tag{1.15}$$

The first layer of the net receives the raw input $\mathbf{x}$, while the rest receive the output of the previous *hidden* layer $\mathbf{h}_{l-1}$. The last layer is responsible for computing the probabilities for each class, and then usually implements a softmax function with which we can estimate $p(c|\mathbf{x})$ and output $\mathbf{c}$ (see 1.3).

You must use the same notation

$$f_L(\mathbf{h}_{L-1}) = S(\mathbf{W}_l\mathbf{h}_{l-1} + \mathbf{b}_l) \tag{1.16}$$

A neural network can be any model that arranges differentiable activation functions into a directed acyclic graph (DAG), mapping input to output. In 1.14 we have showcased one of the simplest models, where the DAG is a chain, which we refer to as a Feed Forward Neural Network or FFNN.

The weight parameters of this DAG can be optimized for a *loss function* $\mathcal{L}$ that can be computed for each sample $(\mathbf{x}, \mathbf{C})$ comparing the output of the model $\mathbf{c}$ to the true label $\mathbf{C}$. We refer to this opmtimization as the training of the model, which is commonly done using gradient descent and backpropagation [31].

Use lowercase c

Despite the Perceptron dates from 1958, the XOR Problem from 1969 and backpropagation algorithm from 1986, it wasn't until recently that neural networks have become a common choice for ML systems.

two

We can see 2 first succesful usages that improved the state of the art. In 2011 [11] proposed a system in the ASR field a system that replaced the common choice of Gaussian Mixture Models by a deep approach, and in 2012 [23] showed a CNN that offered a gap in the error rate (from 26% to 15% in the top-5 task) for the Imagenet classification task with respect to previous models, that were only improving at a rate of aproximately 2% each year.

not necessary this acronym was previously introduced

in ML tasks

I would include the citation at the end of the sentence and I would construct a passive sentence

Two reasons that could be attributed to their recent success are that the internet growth has provided large labeled datasets, which allow the training of bigger models without overfit and the increase in computational capacity given by GPUs, specially to perform matrix and tensor operations.

1.4 Machine Translation

The translation of languages is a problem whose automatization has been in research for decades. Many have been the challenges found and the breakthroughs made, as well as the changes of paradigm seen.

Throughout history we've been able to see optimism when progress was made as well as dissillussion when certain problems saw no way forward on solving. The growth in necessity for systems that automatically translated texts (administrative, commercial, ...) in all areas of a more multi-lingual and connected world has driven the investments in research that allowed to push forwards the limits in the field through continuated efforts.

As in any AI task, we want here to see computers showing the appearence of intelligence, the intelligence required to find how a concept encapsulated in words of one language, can be expressed in words of another language. For a person this task often requires profound knowledge of both languages, unraveling meaning and arranging a good representation of it in words from a complete different set, a different language that is influenced by historical, cultural and social factors.

Thus we can see how this mapping can't be most times completely pure, because there's noise in the encapsulating of concepts and in distiling meaning from their representations, even when the process is done by native speakers.

We can distinguish how historical attemps to build translation systems have been based on gathering knowledge of the languages that are to be translated, with the stablishment of sintactic and semantic rules built into the system. This involved the work of experts in the languages to build these systems.

However, the predeominant and recent trend has been the use of methods that rather rely on data that is available. The model that learns how to perform the task is built up from data. This means that we need big sets of well translated sentences on the two languages that we're considering, and from which the model will (learn to) take decisions for future sentences.

Formal setting

Machine Translation can be seen as a classification task where the set of possible classes is every sentence that can be produce with the vocabulary of our target language. This is an infinite set of possible classes if we consider sentences of any length, or a very big set if we delimit the max and minimum length of a possible sentence to be produced.

We represent a sentence that is to be translated as a vector of $\mathbf{n}$ words $\mathbf{x} = (x_1, x_2, x_3, ..., x_n)$ and its translation as a vector of $\mathbf{m}$ words $\mathbf{y} = (y_1, y_2, y_3, ..., y_m)$. These are the *source* and *target* sentences respectively. If we refer to the prediction made by the system we will use $\mathbf{y}_p$ while if we refer to a given correct translation of sentence $\mathbf{x}$ we will use $\mathbf{y}_c$. Note that in difference to the previous section 1.3.1 we use y instead of c to refer to a classes.

As a supervised classification task, our goal is to result in the $\mathbf{y}$ that maximizes the probability $p(\mathbf{y}|\mathbf{x})$. In this case, result in the best translation possible, the sequence of words $\mathbf{y}^*$ that are in most probability a translation of $\mathbf{x}$. This would be the purest translation of $\mathbf{x}$, the one that matches maximally the concepts that $\mathbf{x}$ encloses. We could define though, that a correct translation $\mathbf{y}_c$ of $\mathbf{x}$ of language s into language t is one that cannot be distinguished from a translation given by a professional human translator.

$$\mathbf{y}_p = \arg\max_{\mathbf{y} \in Y} p(\mathbf{y}|\mathbf{x}) \tag{1.17}$$

$$\mathbf{y}_p = \arg\max_{\mathbf{y}\in Y} p(\mathbf{x}|\mathbf{y})p(\mathbf{y})^1 \tag{1.18}$$

Where Y is the set of possible sentences that can be produced in the language of translation.

1.4.1. Word and Phrase Based Models

We can recall two families of statistical machine translation models that were historically in research and used this decomposition to approximate the probabilities. These were *word-based* models and *phrase-based* models. In both, we can distinguish two parts, the *language* model and the *translation* model, which represent the probabilities $p(\mathbf{y})$ and $p(\mathbf{x}|\mathbf{y})$ respectively from Equation 1.18.

Word based models [8], as the intuition their name might give, were models where the translation of a sentence was achieved by computing a best guess for the translation of each word of the sentence.

Phrase-based models [22], in contrast, treat a sentence as a sequence of different non-overlapping blocks of words, and then is each one of these blocks which receives its translation to the target language as a unit.

The IBM-1 Machine was an example of a word based model. For computing an estimation to $p(\mathbf{x}|\mathbf{y})$, the translation model, it relied mainly on two aspects. First, a way to learn from well translated sentence pairs what is a best translation for each word individually. This was base based on how each word was most times translated in this data, the real world experience supplied to the system.

Second, a way in which the individually translated words needed to be reordered in the target language of consideration. For this end, it introduced an alignment variable, which defined what was the position in the target sentence, for each word of the source sentence. To learn an estimation for this alignment given a sentence to be translated, it used the EM algorithm [12], again, based on the information of the language pair present in the human translated parallel corpora which the system had access to.

For an estimation to $p(\mathbf{y})$, the language model, n-gram models [7] were used. A language model rather than considering the problem of the language pair, captures information about which words of one language are more likely to appear together in natural expressions of it. In other words, it is a reference of which sentences are more probable than others.

This could help the system to reach decisions when it considers between different translations for the next word, also achieving greater familiarity in the result. It is a way to incorporate into the system the structure of the language of which we want to produce sentences, translation sentences in this case.

The inclusion of a language model allows to make direct use of monolingual corpora of the desired language, which is in most cases more widely available, in greater amounts and more easily gathered.

A way one could approximate $p(\mathbf{y})$ would be to think of which is the probability of the next word in a sentence (y_n) given the previous words, and then give the probability for the whole sentence using the chain rule.

$$p(\mathbf{y}) = p(y_n|y_{n-1}, y_{n-2}, ..., y_0)p(y_{n-1}|y_{n-2}, ..., y_0) \times ... \times p(y_1|y_0)p(y_0) \tag{1.19}$$

Given that the number of different possible preceding words grows exponentially, a n-gram model makes the assumption that the word at position i will depend on the context

[1] See 1.4

of only n preceding words, thus taking into account a limited history and achieving computational feaseability.

As for phrase based models, it was log linear models which were a clear representative. Their language model was also estimated using n-grams, whereas the translation model was formed by two parts, a *lexicon translation* model and a *reordering* model. Although we will not dive into an explanation of these constituents, they were the state of the art for the machine translation task before deep neural networks took main presence.

1.5 Neural Machine Translation

As we've seen in section 1.3.2, the use of neural networks for classification tasks started to show results that created performance gaps in comparison to non-deep systems. FFNNs worked for structured data and CNNs worked best for images. As these outcomes appeared each time more present, deep models were also brought to the translation framework.

In this scenario where data was unstructured, in form of sequences of variable length, RNN architectures were the main actors to take part. First, deep learning approaches were used to replace only certain parts of the systems, while the first model using a net for the whole translation pipeline was a RNN with an attention mecanhism [5].

1.5.1. RNN and Translation

Different tasks can be tackled in the scenario of working with sequences, and then different structures for RNNs can be seen. We could want the output to be a vector of labels into which classify an input sentence (for example wheter an statement is negative, positive or neutral) or the inverse problem, given an input in form of a vector, we could expect a sequential output (e.g. image captioning generation). However, we will focus in the problem called sentence translation (language translation in this case) which maps sequence to sequence. (figure to seq2seq rnn).

A Recurrent Neural Network is a neural network which incorporates an internal *hidden* state $\mathbf{h} = (h_1, h_2, ..., h_T)$ where T is the length of the output. The output produced in the model will then depend on this hidden state in addition to the input introduced in the system. The key to working with sequences is that the internal state of the model will be updated word per word along the processing of $\mathbf{x}$.

This means, it will capture information about the preceding words (or future words of the sequence too in a BiRNN [32]) when working at word at position n.

$$p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{y}, \mathbf{h}|\mathbf{x}) \tag{1.20}$$

These models and the later transformer achitecture, model $p(\mathbf{y}|\mathbf{x})$ directly (see 1.17), in contrast to the non-neural approaches described in section 1.4. Even if we denote $\mathbf{n}$ as the length of the input sequence, a RNN expects inputs of variable $\mathbf{n}$, while in a FFNN for example, once the model is defined, the input length remains constant.

Each state h_t could be built as a LSTM unit [17] or a GRU unit [10], that determines which information should be kept from the previous state h_{t-1} and from the upcoming word x_t, to be passed to the next state. If we combine two RNN one building $\mathbf{h}$ from the start of the sentence to the end and the other in inverse order, we would be describing a Bidirectional RNN.

$$h_t^{\rightarrow} = \varphi(\mathbf{W_x}^{\rightarrow} x_t^{\rightarrow} + \mathbf{W_h}^{\rightarrow} h_{t-1}^{\rightarrow} + \mathbf{b}^{\rightarrow}) \tag{1.21}$$

$$h_t^{\leftarrow} = \varphi(\mathbf{W_x}^{\leftarrow} x_t^{\leftarrow} + \mathbf{W_h}^{\leftarrow} h_{t+1}^{\leftarrow} + \mathbf{b}^{\leftarrow}) \tag{1.22}$$

$$h_t = [h_t^{\rightarrow}, h_t^{\leftarrow}] \tag{1.23}$$

Where φ would be a differentiable nonlinear function. The state h_t represents then information the past (words prior to x_n) as well as information of the future.

When we reach the final word of the sentence ($t = T$), $\mathbf{h}$ has been built upon preceding states h_{t-1}:h_1 and all words have been read. Thus we can see $\mathbf{h}$ as a representation of the information the sentence contained.

This addresses one basic problem in the translation pipeline. How to represent sentences that are verbal human compositions, in a numerical computer-redeable arrangement.

Since we are in a deep learning framework, which information is extracted from the words will depend on the values of weight matrices as we see in Eq 1.21, 1.22 and the task of discovering which is a proper set of weights is then let to the network to be optimized through backpropagation.

Once this representation of the input data is produced by the model (being the output of this network the last hidden state itself), we could use another RNN, that learns the inverse problem. Receveing $\mathbf{h}$ as input, produce word per word a sentence, but in the desired language of translation. The internal state $\mathbf{s}$ of this *decoding* RNN would be updated with the preceding state together with the word the system just previously produced until the end of sentence is brought out.

$$s_t = \varphi(y_{t-1}, s_{t-1}) \tag{1.24}$$

This scheme where input information is encoded in a subspace and then decoded into the solution of the problem, can be seen in general under the name encoder-decoder, where each module isn't necessarily implemented by a RNN nor a NN.

As mentioned in the beginning of this section, it was the model used in [5], with a BiRNN as encoder and a RNN as the decoder.

This was the first standalone neural model for machine translation, i.e. the first translation system where the whole model was a single neural network (3 RNN working together) and the whole trained together at a time.

It included though an element of main relevance, an *attention mechanism* that modified the update step of the internal state in the decoder (see 1.24) RNN to depended on an extra *context* vector $\mathbf{c_t}$.

$$s_t = \varphi(y_{t-1}, s_{t-1}, \mathbf{c_t}) \tag{1.25}$$

Attention mechanism

Attention is a technique that (in a RNN) allows an state h_t to receive information directly from any other state h_y in the network. If we see h_t as the representative of meaning when looking a sentence at position t, with attention we build an structure in the network that allows it to learn which states are more dependant on each other, and thus should be most taken into account when building each other state.

We could imagine an scenario where the meaning of a word in a sentence is understood in relation to other distant words. Attention would allow a connection between these, that otherwise might have been lost along all updates of $\mathbf{h}$. Having this extra information allows better representations, leading to better results. And are the better results during training of the net which guide the building (or discovery) of these dependencies.

Following [27], attention can be described as a dictionary lookup where for a query $\mathbf{q}$ and a set of keys $\mathbf{k_i} \in \mathbf{K}$, we compute a combination of the values $\mathbf{v_i} \in \mathbf{V}$.

$$Attn(\mathbf{q}, (\mathbf{K}, \mathbf{V})) = \sum_i \alpha_i(\mathbf{q}, \mathbf{K})\mathbf{v_i} \tag{1.26}$$

The strength of the connection between the query $\mathbf{q}$ and the key $\mathbf{k_i}$ is modeled by the *attention weight* $0 \leq \alpha_i(\mathbf{q}, \mathbf{K}) \leq 1$

$$\alpha_i(q, K) = S_i(a(\mathbf{q}, \mathbf{k_1}), ..., a(q, \mathbf{k_n})) = \frac{\exp(a(\mathbf{q}, \mathbf{k_i})}{\sum_j^n \exp(a(\mathbf{q}, \mathbf{k_j}))} \tag{1.27}$$

Where S is the softmax function and the definition of a gives name to what is called *scaled dot product attention*

$$a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q^T k}}{\sqrt{d}} \tag{1.28}$$

Where d is the length of both the query and key.

In the context of a RNN with hidden states representing the meaning of words, attention could compute the dependency of hidden state h_t to each other state h_i.

$$Attn(h_t, (\mathbf{h}, \mathbf{h})) = \sum_{i=1}^{T} \alpha_i(h_t, \mathbf{h})h_i \tag{1.29}$$

Thus attention when focusing on a sentence position will be a combination of the hidden states, where the states will have a bigger or smaller weight depending on how their dependency to the word at that position is relevant for producing better results. We allow the network setting low weights for some states (ignoring parts of the sentence for this word) or emphasizing the meaning other parts (high attention weight values).

It has been observed that systems that didn't use attention provided worse results, specially when working with long sentences where these distant dependencies are hard to handle by a native RNN in its hidden state.

In the neural network system for translation that we have previously referenced [5], each state $\mathbf{s_i}$ of the decoder attended not its own hidden states but the hidden states of the encoder. This allowed to dinamically focus on parts of the source sentence upon each production of a word for the target sentence.

If we recall Equation 1.25 we can now define the context vector $\mathbf{c_t}$.

$$\mathbf{c_t} = \sum_{i=1}^{T} \alpha_i(s_{t-1}, \mathbf{h})h_i \tag{1.30}$$

The attention weights can be seen acting as an alignment of target words to source words, ~~hence the title of the article.~~

1.5.2. The Transformer Model

If the first neural standalone model was released in 2016, it wasn't long before the Transformer model was published in 2017. If models using attention were showing state of the art results in the area, the transformer based its architecture fully on it.

Also following an encoder-decoder scheme, attention is used in both parts, learning the representation of data during encoding as well as producing each resulting word in

the decoder. Given the wide employment of the model, there are many good references that explain it [39], [27], [40], [20], in addition to its many variations.

Since it is the model at the core of the systems developed in this work, we will describe the architecture and its fundamental features, so the reader can have a realistical grasp on the tasks carried out along the project.

Self Attention Architecture

From a zoomed out view, the Transformer is built of a stack of encoding units on top of each other that pass their information to another stack of decoding units at the top of which output words are produced. The model can be understood understanding these two kinds of units.

When a sentence is supplied to the system, we don't have now a hidden state, so instead of processing it word per word as in a RNN, attention is performed to each word with respect to each other word of the sentence hence called *self* attention. This becomes the new representation of the word, that has gathered valuable context from the other words.

The new representations are passed to the next encoding unit (after extra process as seen in fig (encoding unit)), that will perform the same operations. This change in architecture allows high paralelization that results in training and inference speed improvements.

If input word is $\mathbf{x_i}$ and output word $\mathbf{o_i}$ after one attention layer,

$$\mathbf{o_i} = Attn(\mathbf{x_i}, (\mathbf{x_1}, ..., \mathbf{x_n}), (\mathbf{x_1}, ..., \mathbf{x_n})) \tag{1.31}$$

In practice attention is carried out through matrix operations, all queries arranged in matrix $\mathbf{Q}$. We keep using Scaled Dot Product Attention (see 1.28).

$$Attn(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = S(\frac{\mathbf{Q}\mathbf{K}^\mathbf{T}}{\sqrt{d}})\mathbf{V} \tag{1.32}$$

Multi Head Self Attention

In addition to this new base idea for learning the representations for the words, we don't only have one set of attention weights per word but ~~a number~~ of them. This means that multiple dependencies or similarities can be captured for a word with respect to all other words in the sentence, with a different set of weights specialized for each one of these dependencies.

For example, if we consider the word 'it' in a sentence, one set of weights could focus on which subject this pronoun refer to, while other attention *head* could focus on adjectives that are describing it.

$$\mathbf{head_i} = Attn(\mathbf{Q}\mathbf{W_i^q}, \mathbf{K}\mathbf{W_i^k}, \mathbf{V}\mathbf{W_i^v}) \tag{1.33}$$

Where queries, keys and values have been linearly projected to other subspaces by the trainable matrices $\mathbf{W_i^q}, \mathbf{W_i^k}, \mathbf{W_i^v}$.

Heads are concatenated, and to restore the original dimension, multiplied by $\mathbf{W^o}$.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\mathbf{head_1}, ...\mathbf{head_h})\mathbf{W^o} \tag{1.34}$$

Multihead attention is applied in the encoder stack as we have described, while in a decoding unit, it is applied in two different ways.

For an output word being produced, the aim is to receive information from the source sentence words as well as the already translated output words. This is why, words of the target sentence go through a first layer of self *masked* multi head attention.

This restricts the mechanism of attending words that haven't been yet translated. Even if during training time all correct translations are known, it isn't the case once the system is in production. That's why the model has to learn to translate without relying on this information.

Successively after this layer, there's a layer of multi head attention that attends source sentence information. Queries are the current representation for output words $\mathbf{y_i}$ and keys and values are each of the positions of the output of the encoding stack. (see fig)

So in a decoder unit, better representations are learnt from previous translated words, and then the information of the encoding part is gathered too. These operations are repeated in chain for each unit until the top of the decoding stack is reached.

Layer Normalization and Residual Connections

As we see in figures (,) in addition to attention layers, FFNNs (1.14) are used to build the units in this case with ReLU activation functions [1].

After both, FFNN or attention, *Layer normalization* [4] and *Residual connections* [16] are performed.

These are common techniques in Neural Networks that help the optimization process of the net. Layer normalization aims to prevent the problem of vanishing and exploding gradients (see [27] section 13.4.2), while residual connections allow the backward flow of gradients during backpropagation even when gradient vanishing occured in parts of the net. They allow to train deeper models with much bigger number of layers.

Residual connections allow the input of a layer to *skip* all the transformations F that the layer perform to x, thus add x at the ~~end~~ output of the layer.

$$F'(x) = F(x) + x \tag{1.35}$$

$F'(x)$ being the output of the residual block.

Layer normalization computes the mean and variance across outpus of hidden units of a layer before the activaction function.

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \quad ; \quad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \tag{1.36}$$

$$a_i^l = w_i^{l^T} h^l \quad ; \quad h_i^{l+1} = f(a_i^l + b_i^l) \tag{1.37}$$

Where w_i^l are the weights of hidden unit i and h_l, the inputs receveid from the previous hidden layer l. Then the varianze divides and the average is substracted.

$$\hat{a}_i^l = \frac{a_i^l}{\sigma^l} \tag{1.38}$$

$$\text{LayerNorm}(a_i^l) = \gamma \hat{a}_i^l + \beta \tag{1.39}$$

γ and β being trainable weights that allow the net to undo these changes partially or completely favouring the optimization.

Input representation and Positional encoding

If we recall RNNs, words were processed sequentially, and hidden states updated accordingly in a sequential fashion, thus capturing information about the position occupied by each word in the sentence.

This is not the case in the Transformer since attention is invariant to the ordering of the words. To provide the model with this information, a **positional embedding** is combined with the **word embedding**.

When a sentence goes into the model, prior to transforming it through all attention layers, we need to change from a natural language representation to a numerical one, and so each word is first transformed to a one-hot vector, that represents its corresponding index in the vocabulary. So a sentence would be represented in a matrix of size ($n \times vocabulary\_size$), where n is the number of words.

After this, it is multiplied by a learnt embedding matrice $\mathbf{E_e}$, of size ($vocabulary\_size \times d$), where d is a parameter of the model. Words are projected to this dimension, ending with a representation of a sentence in dimension ($n \times d$). This is the word embedding, **WE**.

$$\mathbf{WE} = \mathbf{x}^{2}\mathbf{E_e} \tag{1.40}$$

Now, to account for position information, we build a matrix also of dimension ($n \times d$) where each row represents the position of a word in a d-dimensional vector. This matrix could also be learnt, but the Transformer uses a fix encoding instead, that is given by two sinusoidal functions.

$$\mathbf{PE}_{(pos,2i)} = sin(pos/10000^{2i/d}) \tag{1.41}$$

$$\mathbf{PE}_{(pos,2i+1)} = cos(pos/10000^{2i/d}) \tag{1.42}$$

Where pos is the position of the word in the sentence,and the corresponding row of the **PE** matrix, and i each index of this row.

$$Embedding(\mathbf{x}) = \mathbf{WE} + \mathbf{PE} \tag{1.43}$$

This choice for the positional encoding provided no performance decrement compared to a learnt embedding, while allowed the model to extrapolate to sentence lenghts that were unseen during training.

Exactly the same process is applied to source words on the encoding as to words that enter the decoder stack ($y_{t-1} \forall t \in [1, T]$ produced by the decoder in previous instants).

The Transformer model also uses a technique called Weight Tying [28] where a single matrix **E** is used for the encoding as for the decoding, and also for the output embedding matrix($\mathbf{E_o} = \mathbf{E^T}$), that transforms the output of the last hidden layer in the encoder into probabilities for the next word when given to the softmax layer after a learnt linear transformation.

Transformer architectures have shown success with state of the art results in machine translation as well as in a wide range of NLP tasks. We can name language modeling, document summarization, sentiment analysis, text generation, text paraphrasing, reading comprehension or question answering. GPT-3 [9], BERT [13], or T5 [29] are Transformer architectures. Whenever we think of a task that involves working with language or see a translator offering good results in production, there's probably a Transformer operating in behind.

[2] The one-hot representation of the input sentence

Recent results of the 2022 IWSLT competition [2] show the wide and present use of the Transformer model for machine translation tasks.

1.6 Evaluation

1.7 Framework of this work

1.8 Document structure

????? ????????????? ????????????? ????????????? ????????????? ?????????????

CHAPTER 2
Data

In this chapter we will describe the data that we will be using in our systems, as well as the different processing techniques that this data will undergo prior to any use of it by our models.

2.1 Datasets

As we have described in chapter I, all learning and thus all performance that our systmes will achieve will be based on the real world experience that we make them available with.

In our Machine Translation scenario where data consists of sentences paired with their translations, the model will assume that the translation that we give is correct. Thus if we offer the system with incorrect translations, this will be reflected in the system performance, since predictions will be based on these false truths. With this we can understand how the quality of the data is important in building our systems.

The quantity of data is another main factor with what respect to system results. The Transformer is a big architecture that allows us to train with datasets consisting of millions of sentence pairs. Systems offering state of the art performance are usually trained using very big datasets, that allow us to train without overfit, these big models that have billions of parameters. For example, GPT-3 [9] is trained with a dataset consisting of aproximately 499 billions of words.

This opens a compromise, since there is need for big amounts of data, but manual production is highly resource consuming for many tasks. It is so specially in MT where we need paired sentence translations and we could potentially want to translate between many different language pairs. In this labour, there are many collaborative works that have achieved the formation of big public parallel datasets. We can refer to OPUS [3] as a work that tries to gather in a single open site, big part of the available datasets for machine translation.

2.1.1. Training Dataset

Our training dataset has been formed with the purpose of training a general domain translation model i.e. a model built for translating sentences of the English language without making assumptions of their contents. In this way, we have gathered open sources that contain sentences of very different contexts, with no fixation on any in particular. We can see a summary of the dataset in figure (training set portions).

We will see in Chapter 5 how having prior knowledge about the phrases that our model will translate, can be used for building a model that performs better for this data, with a process that encloses a set of different techniques known as *domain adaptation*.

We count up to 309 million sentence pairs in our dataset, which makes a total of $5.6 \cdot 10^9$ words for the English language and $6.3 \cdot 10^9$ words for the French language.

A portion of the datasets that we use are built crawling parallel text resources from the web. In this task, automatic tools to process and cleaning the data are determining to produce sets of good quality. Of this kind we have:

- CommonCrawl[1], CCAligned [14] and ParaCrawl [6] from different sources of the web.
- WikiMatrix [33] and WikiMedia [37] sourced from Wikipedia.
- Giga Fr-En [38] crawled from Canadian and European Union sources for the WMT2010 Competition [2].

The rest of the parallel sets that form our training data can be classified in a group of sources that have not been gathered from the web:

- The United Nations Parallel Corpus (UNPC [41]) from manual translations of United Nations documents ranging from 1999 to 2014.
- EUBookshop [35] from publications of European institutions.
- Europarl [19] and Europarl-ST [18] from the European Parliament.
- DGT-TM [36] published by the same DGT (Directorate-General for Translation), a department in the European Commission and one of the biggest translation services in the world.
- News Commentary [38] a collection of News Commentaries provided by WMT and sourced from CASMACAT[3].

2.1.2. Development and Test Datasets from WMT

The WMT Conference on Machine Translation (originally Workshop on Machine Translation) is a main event in the field of Machine Translation Research. The conference is held anually, where universities, research laboratories and technology companies participate pushing performance boundaries of the systems for a set of proposed machine translation tasks. We have decided to evaluate the performance of our system with the tests sets of the WMT13[4] and WMT14[5] competitions. These are general domain data of high quality.

Since development and test sets in machine translation are relatively much smaller than training sets (~~of~~ in the orders of thousands of sentences) they can be reviewed in detail for adequate translations, assuring to be works useful to contrast the quality of our translations. We use WMT13 and WMT14 respectively as the principal development and test sets for measuring the performance of our systems. We will see in Chapter 5 the use of a different development and test set to evaluate the performance on a specific domain.

[1] https://commoncrawl.org/the-data/
[2] https://www.statmt.org/wmt10/translation-task.html
[3] http://www.casmacat.eu/corpus/news-commentary.html
[4] https://www.statmt.org/wmt13/translation-task.html
[5] https://www.statmt.org/wmt14/translation-task.html

2.2 Preprocessing

Data preprocessing is a common step in NMT systems prior to feeding our models with this data either for training or for infering the translations. It works improving the quality of the results as well as the computational efficiency and feasibility of the models. We will now describe the different techniques that we used when developing our systems.

2.2.1. Tokenize and Truecasing

To understand these steps we need to understand that our systems work with a vocabulary of fixed and limited size i.e. a limitied number of words that will be recognized from the English language and a limited number of words that will be possibly produced in French (we will see in section 2.2.2 techniques to pragmatically overcome this limitations even if keeping the vocabulary fixed and limited).

If we refer to section 1.5.2 we see that the size of the one-hot vectors representing each word of an input sentence depends directly on the vocabulary size of the source language, while the size of the final softmax layer of the model(reference to transformer figure) that accounts for probabilities for each next word, depends on the size of the output vocabulary. In this way, reducing vocabulary sizes translates in reducing memory consumptions and increasing computational efficiency.

If we considered each sequence of symbols that is separated by blank spaces to be a word of our vocabulary, we would encounter that we could have many different word instances that represent the same word. Consider for example *Okey! Okey... Okey, Okey?*. These four would be considered individual words of our vocabulary, which would grow extensively big. We leave then the task of deciding what constitutes a word to a *tokenizer*, that in this case would separate the word *Okey* from punctuation, and treat each part as an indidual *token* of the vocabulary. See figure (tokenization) for an example of *tokenization* performed in our systems.

Remember to spell-check *

An additional step that reduces the size of our vocabularies consists in dealing with upper and lowercase letters. Instead of keeping diverse versions for a word (consider for example *ATTENTION*, *Attention*, *attention*), we train a *truecasing* model that predicts which case should be kept based on the frequency of each case of each word in our data. A simpler approach to this issue would be to lowercase all words, but more linguistical information would be lost.

In our work we use the Moses tokenizer and truecasing [21]. We can see an example of their use in figure (tok and true example).

2.2.2. Subword Segmentation

The objective of these preprocessing techniques is to increase translation quality by allowing to recognize and produce words that are not present in our vocabularies. We want to mimic what would be having a system that works with an open vocabulary but still keeping it of fixed size.

The idea of these techniques is to break the vocabulary into subword units, which joined together can form the words that were present in the original vocabulary as well as new words that weren't present. The two techniques that we will describe are not to be used complementarily but rather exchangeably.

Byte Pair Encoding

Byte Pair Encoding or BPE [34] [15] works by first splitting words to individual characters. Then we count which pair of consecutive characters is most frequent along all training data, and all instances of this pair are merged into a single unit. This merge operation is saved (the information of which pair of characters constituted the merge) and the process continues, computing the most frequent pair each time, joining the units accordingly and saving the merge pair that corresponded to the maximum count. The process ends when a number of maximum different pairs (merge operations) is reached, and this information is stored. This number is the only parameter of the model and it is to be defined.

This constitutes the training of the BPE model, and once the merge operations are learnt, we can apply them to new data (during inference) in addition to the training data, although to form our vocabularies we would apply it to the training set (after tokenization and truecase). We would split again the text into single characters and apply the stored merge operations in order. The resulting subword units would form the vocabulary of the model. Units that represent the end of a word are followed by the string <\w> and subwords didn't reach to be rejoined into a complete word are followed by the symbols @@, so that original words can be restored after translation.

We can see an example of the result when applied BPE to one of the training sentences in figure (bpe example).Truecasing and Tokenization have been applied before BPE.

Sentence Piece

In contrast to BPE, Sentence Piece [24] doesn't expect a tokenized input. The main feature in difference is that it treats whitespaces as one character more, and by replacing it by the symbol _, the tokenization is learnt on the way, as a result of the process of learning the merge operations that will define the subword units. This allows to have a language-independent tokenization.

For example, most tokens in European languages might be separated by whitespaces, while for example in Japanese, different tokens are joined together without whitespaces between them. Thus there's usually need of using language-dependent tokenizers. In addition to this, in common tokenizers, there are operations that need specific rules in order to be reversed back during detokenizing. The information that there is no whitespace between the end of a word and a punctuation sign, is lost while tokenizing, but in SPM it's not, since there would be no _ between these characters. With this, SPM offer what they refer as a *Lossless* tokenization, i.e. detokenization being the inverse operation of tokenization.

Sentence Piece comprises four main modules, a *Normalizer*, a *Trainer* an *Encoder* and *Decoder*. The normalizer handles semantically-equivalent characters and unites them into a single representation. The trainer learns from our training set the merge operations that we need to transform text into subword units. In difference to BPE, instead having a definite number of operations, we define the final desired size of the vocabulary, and the number of merge operations is calculated accordingly to give this result. The encoder applies the learnt operations to a given text, achieving subword segmentation as well as tokenization and the decoder is simply the inverse operation.

We can see an example application of SentencePiece in figure (spm example) when we have first applied a Truecasing preprocess.

2.2.3. Filtering

The aim of data filtering is to improve the quality of the data that we feed our models for training. The different techniques try to reduce the number of sentences that contain translations that are undesirable to have in our data and ~~from~~ which to learn. There are many different sources of noise that can decrease the quality and performance. One could encounter text that is in different languages than ~~the one source of translation~~ (even with different alphabets e.g Chinese, Russian,...), characters represented in an incorrect format, html text, incorrect uses of language, mispelled words, gramaticall errors, too long or too short translations, etc. Specially when using resources crawled from the web, the need of filtering is higher to reduce this noise and if we are not cautious, adding more data to our systems could ~~result in the contrary of the desired~~ effect.

For these reasons, the parallel corpora ParaCrawl which constitutes the majority of our training sentences and which was crawled from the web, got filtered with two tools, bitextor-bicleaner[6] and bitextor-bifixer[7]. In addition, ~~as we will see in Chapter 4~~ we applied in different versions of our systems two different extra types of filtering techniques, these are *language identification* and *source-to-target ratio*.

As the intuition given by its name, the language identification filter tries to recognize when a language different than the source or target language being used (in the corresponding source or target sentences), and then remove these sentences in both parts of the corpora. Source-to-target ratio simply detects when the length (in number of words) ratio between source and ~~translation~~ surpasses a given threshold. We define a limit where the translation is excessively short or long to be considered a good translation and we desire to remove it from the data.

[6] https://github.com/bitextor/bicleaner
[7] https://github.com/bitextor/bifixer

CHAPTER 3
CERN News

3.1 Crawling

3.2 Alignment

3.3 Post-Processing

CHAPTER 4
Offline NMT Systems

4.1 Fairseq

4.2 System in Production

4.3 Results

4.3.1. v1 Baseline

4.3.2. v2 Language ID

4.3.3. v3 Sentence Piece

CHAPTER 5
Domain Adaptation

5.1 Finetuning

5.1.1. CERN News

5.1.2. CERN Document Server backtranslations

5.2 Training with backtranslations

5.2.1. Results: v4 Backtranslations

CHAPTER 6
Simultaneous NMT

6.1 Online Scenario

6.2 Evaluation

6.3 Results

6.3.1. v4 Multi-Path-Wait-K

CHAPTER 7
Conclusions

????? ????????????? ????????????? ????????????? ????????????? ?????????????

Table 7.1: En->Fr Bleu scores for WMT, CERN and CERNnews evaluation sets

System	wmt13	wmt14	CERN-dev	CERN-test	CN21	CN22
prod	**34.8**	40.8	**65.4**	**67.0**	37.2	37.7
v1	32.1	39.1	54.2	58.5	-	-
v2	32.6	39.4	53.9	57.7	-	-
v3	34.0	**41.0**	53.8	58.0	38.3	38.7
v4	34.0	40.9	53.2	57.4	**38.6**	**38.8**
v3FTk100	-	-	-	-	42.0	**43.1**
v4FTk100	-	-	-	-	**42.3**	42.9

Bibliography

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018.

[2] Antonios Anastasopoulos, Loïc Barrault, Luisa Bentivogli, Marcely Zanon Boito, Ondřej Bojar, Roldano Cattoni, Anna Currey, Georgiana Dinu, Kevin Duh, Maha Elbayad, Clara Emmanuel, Yannick Estève, Marcello Federico, Christian Federmann, Souhir Gahbiche, Hongyu Gong, Roman Grundkiewicz, Barry Haddow, Benjamin Hsu, Dávid Javorský, Věra Kloudová, Surafel Lakew, Xutai Ma, Prashant Mathur, Paul McNamee, Kenton Murray, Maria Nădejde, Satoshi Nakamura, Matteo Negri, Jan Niehues, Xing Niu, John Ortega, Juan Pino, Elizabeth Salesky, Jiatong Shi, Matthias Sperber, Sebastian Stüker, Katsuhito Sudoh, Marco Turchi, Yogesh Virkar, Alexander Waibel, Changhan Wang, and Shinji Watanabe. Findings of the IWSLT 2022 evaluation campaign. In *Proceedings of the 19th International Conference on Spoken Language Translation (IWSLT 2022)*, pages 98–157, Dublin, Ireland (in-person and online), May 2022. Association for Computational Linguistics.

[3] Mikko Aulamo and Jörg Tiedemann. The OPUS resource repository: An open package for creating parallel corpora and machine translation services. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 389–394, Turku, Finland, September–October 2019. Linköping University Electronic Press.

[4] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[6] Marta Bañón, Pinzhen Chen, Barry Haddow, Kenneth Heafield, Hieu Hoang, Miquel Esplà-Gomis, Mikel L. Forcada, Amir Kamran, Faheem Kirefu, Philipp Koehn, Sergio Ortiz Rojas, Leopoldo Pla Sempere, Gema Ramírez-Sánchez, Elsa Sarrías, Marek Strelec, Brian Thompson, William Waites, Dion Wiggins, and Jaume Zaragoza. ParaCrawl: Web-scale acquisition of parallel corpora. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4555–4567, Online, July 2020. Association for Computational Linguistics.

[7] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8):1157–1166, 1997. Papers from the Sixth International World Wide Web Conference.

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

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell,

Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[10] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[11] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Large vocabulary continuous speech recognition with context-dependent dbn-hmms. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4688–4691. IEEE, 2011.

[12] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39:1–38, 1977.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[14] Ahmed El-Kishky, Vishrav Chaudhary, Francisco Guzmán, and Philipp Koehn. CCAligned: A massive collection of cross-lingual web-document pairs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5960–5969, Online, November 2020. Association for Computational Linguistics.

[15] Philip Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, feb 1994.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] Javier Iranzo-Sánchez, Joan Albert Silvestre-Cerdà, Javier Jorge, Nahuel Roselló, Adrià Giménez, Albert Sanchís, Jorge Civera, and Alfons Juan. Europarl-st: A multilingual corpus for speech translation of parliamentary debates. *CoRR*, abs/1911.03167, 2019.

[19] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit X: Papers*, pages 79–86, Phuket, Thailand, September 13-15 2005.

[20] Philipp Koehn. *Neural machine translation*. Cambridge University Press, 2020.

[21] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[22] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[24] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, 2018.

[25] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[26] Tom M. Mitchell. *Machine Learning*. New York: McGraw Hill, 1997.

[27] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[28] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain, April 2017. Association for Computational Linguistics.

[29] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019.

[30] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

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

[32] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[33] Holger Schwenk, Vishrav Chaudhary, Shuo Sun, Hongyu Gong, and Francisco Guzmán. Wikimatrix: Mining 135m parallel sentences in 1620 language pairs from wikipedia. *CoRR*, abs/1907.05791, 2019.

[34] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909, 2015.

[35] Raivis Skadiņš, Jörg Tiedemann, Roberts Rozis, and Daiga Deksne. Billions of parallel words for free: Building and using the EU bookshop corpus. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1850–1855, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA).

[36] Ralf Steinberger, Andreas Eisele, Szymon Klocek, Spyridon Pilos, and Patrick Schlüter. DGT-TM: A freely available translation memory in 22 languages. *CoRR*, abs/1309.5226, 2013.

[37] Jörg Tiedemann. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA).

[38] Jörg Tiedemann. Parallel data, tools and interfaces in opus. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA).

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[40] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[41] Michał Ziemski, Marcin Junczys-Dowmunt, and Bruno Pouliquen. The United Nations parallel corpus v1.0. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3530–3534, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).