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

Contents

Contents v
List of Figures vii
List of Tables vii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Goals . . . 1
- 1.3 Articial Intelligence and Machine Learning . . . 1
  - 1.3.1 Artificial Inteligence . . . 1
  - 1.3.2 Machine Learning . . . 2
  - 1.3.3 Supervised learning . . . 2
  - 1.3.4 Clasification . . . 3
  - 1.3.5 Deep learning . . . 4
- 1.4 Machine Translation . . . 5
- 1.5 Neural Machine Translation . . . 7
  - 1.5.1 The Transformer Model . . . 9
- 1.6 Framework of this work . . . 12
- 1.7 Document structure . . . 12

2 Data 13
- 2.1 Preprocessing . . . 13
  - 2.1.1 Tokenize . . . 13
  - 2.1.2 Truecase . . . 13
  - 2.1.3 Byte Pair Encoding . . . 13
- 2.2 Filtering . . . 13
  - 2.2.1 Sentence length ratio . . . 13
  - 2.2.2 Language ID . . . 13
- 2.3 Datasets . . . 13
  - 2.3.1 Europarl-ST . . . 13
  - 2.3.2 CERN . . . 13
  - 2.3.3 WMT . . . 13
  - 2.3.4 Polimedia . . . 13
  - 2.3.5 CERN-News . . . 13

3 Europarl-ST 15
- 3.1 ?? ???? ???? ? ?? ?? . . . 15

4 Conclusions 17

Bibliography 19

Appendices

A Configuració del sistema 21
- A.1 Fase d'inicialització . . . 21
- A.2 Identificació de dispositius . . . 21

B ??? ???????????? ???? 23

List of Figures

List of Tables

1.1 En->Fr Bleu scores for WMT, CERN and CERNnews evaluation sets . . . 12

CHAPTER 1
Introduction

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

1.1 Motivation

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

1.2 Goals

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

1.3 Articial Intelligence and Machine Learning

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

1.3.1. Artificial Inteligence

Artificial inteligence is built in the ground 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.

we've presentiated big interest if computers can run instructions yielding human actions, because they can do them for us, but we can see many examples of systems that work in tasks not expected for human intelligence. predictions, news about reinforcement learning in nuclear fusion, system that uses ai to allow talking (reading mouth movements)

kinds of AI 2 very differentiated lines / branches rules/expert knowledge based, chatbots, when you call to a certain phone line,

ai subset ml subset dl figure

it's clear how deep ai is sinked in hour society, and how further it will, and it's not possible to talk about all these effects without mentioning machine learning

1.3.2. Machine Learning

a bit more of introductin here. What is machine learning. sounds appealing these 2 words together, machines learning, but what are we refering to

"learning" learning from data all learning is stored in "weights", which is synonim for structured storage it isn't learning at all, we have to keep in mind that it is just appearence of learning probabilistically weights at first bad answers, after "seeing" data, better answers, there comes the learning (or training)

in the end it's a math model, a function, materialized in a big physicical arena of transistors, that probabilistically will yield an asnwer, a value, more or less "right". With "right" being what we humans decided it's the ultimately truth cancer or no cancer, should we turn, stop, or straight? which are the categories of persons that exist? how will this protein twist? how would an expert in 2 languages express this sequence of concepts in both?

each problem requiring very different weight keeping structure, very different processes for discovering those weights. aproximating the ideal weights that would always yield correct answers. a function being optimized (trained), and this optimization being data driven how many days are left for our death? -> link to page that estimates this hahaha

clearly, there is always information missing in our systems, information that isn't accessible, though there are lots lots of information present in physical-cpu-readable systems that can allow these systems to answer questions much further away than human levels the problem is always to create a system that given a limited amount of information (although very big) yields useful answers taking into account that it will incur in falsities

as stated in -KM, TM,

"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"

Here our function is optimized from data (E) to answer questions (T) these being more or less correct (P) lots lots of engineering work, research, discovering better structures, parameters and methods allowing to keep better weights (learning) for better performance (more useful answers) to an specific task (question) from the experience (data). How this data is obtained and processed to be fed into the function to favor the learning is a complete field of research itself as we will see further in the reading. Research that doesn't seem to be lowering any time soon. learning from the structure of data itself (unsupervised learning) -> revisar algo d'aci (heuristicts to the truth distribution of data) reference to famous algorithm of K-means a main difference to supervised learning, in unsupervised learning we are not given a set of questions and correct answers we are just given the set of answers, from which we want to extract some useful information

1.3.3. Supervised learning

learning from high amounts of given questions and truth answers probabilistic

common task is classification many many problems can be viewed as a classification task. ALL neural networks are classification models (nooooooooo hahahah)

hmm, maybe there are better examples word in one language and a context -> to which word gets mapped?

1.3.4. Clasification

=== pattern recognition classification most common task in supervised learning probabilistic. Aço va abans de classificacio?

bayes classifier Data or experience to learn from comes as a pair (x,y), x being the task/question that we are to solve, y being the solution to the task. The correct answer.

Thus, we cannot lose sight however, that the usefulness of a system, our ultimate goal, always comes in answering questions for which there is uncertainty. Inputs x for which the correct solution (y) isn't known.

We're talking about inference here. A model that is able to generalize. Being able to map (classify) a solution to a question for which the answer has not been seen.

Hoping that from the experience "learnt", the system is able to offer answers that are most possibly truth.(labels y to which input feature vectors x get classified). Thus the goal always being to construct a system where false answers answers are minimized. In other words, solutions to tasks *T* so that performance *P* is maximized.

p(error | x) = 1 - p(y=ytruth | x)

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

being y unknown, we can decide to pick the label that minimizes the error. This is the bayes classifier.

min p(error | x) = 1 - arg max p(y | x)

here is where bayes theorem comes

this is called minimum risk decision note that if we knew the true distribution p(y | x) for every x, we would have an optimal performance, an oracle, since x would always be mapped to the most probable y, thus other asigments incurring more errors. (however, this is a fanciful idea.)

We could thus understand ml as a way where we're trying to build a model that tries to aproximate better these distributions, using for such task the knowledge available to our system. (we can many times see the use of bayes rule to deduce an expression easier to approximate arg max p(y | x) = argmax p(x | y)*p(y))

consider the case where we had only 2 possible outcomes. $y \in 0, 1. just 2 possible answers. a question could be$ $Rd), is it a virginica flower or isn't?$

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 w. These weights, would have been learnt (thus learning is just computation of weights under an optimization process of a defined error(penalization) function), after considering many flowers, of which we have measured their attributes, and humanly judged their type.

p(y | x;) = Ber(y | (wTx + b)) 1*downpage sigmoid and theta explanation

we would then predict flower x to be virginica (y=1), if p(y=1 | x) > p(y=0 | x), formally f(x) = wx +b, defining a decision boundary. a line in 2d space, a plane in 3d space and an hyperplane in ND space this is builds a linear classifier. if (iris flowers) can be classified without error uner a linear decision boundary, data is said to be linearly separable we will come back to this problem later, where neural networks come as a solution to discriminate in non-linearly separable data

similarly, we could extend our problem to allow N classes, now estimating the set of possible outcomes with a categorical distribution p(yjx;) = Cat(yjS(Wx + b)) and this still being several linear decision boundaries, or space separated in class regions

better data, better models, better methods, better answers

perceptron algorithm

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

we are going to describe a process which was a fundamental stone or precursor of most nowadays spreaded "learnings" (neural networks), which focuses directly in computing on of this separating hyperplanes, which translates in the end, in finding a good set of weights in the search space of their values, because this hyperplane will be defined by those weights (electrons in millions of capacitors (memory)) dates of year 58 described as a form of actually learning. learning process, optimization process. computing the weights that allow us to discriminate between classes afterwards so we are still talking about a linear classifier, in the scenario of 2 classes f(xn;) = I (wTxn + b > 0)

The basic idea is to start with random weights, and then iteratively update them whenever the model makes a prediction mistake. More precisely, we update the weights using wt+1 = wt t($^y nyn)xnwhere(xn;yn)isthelabeledexamplesampledatiterationt, andtisthelearningrateorste$
$1andy^n = 0, wehavewt+1 = wt+xn, andifyn = 0andy^n = 1, wehavewt+1 = wtxn.itisprooventhatthemetho$

1.3.5. Deep learning

in the models described the mapping was lineal. wx

Multilayer-perceptron XOR problem a classic example of a problem perceptron cannot solve. data which isn't linearly separable. we want to reperesent a function that computes the exclusive OR so given x ∈ $(0,0),(0,1),(1,0),(1,1), ymust = 1onlyifoneoftheinputsisone$

clearly, this scenario isn't linearly separable (figure) no line or plane can be drawn, so that samples (x) get correctly classified

however, with a perceptron, we can model an OR function, and with another one, also an AND function, so we could send our sample x to the OR and the AND. As a result, we would have transformed our input, to another possible intermediate state basis(x) ∈ $(0,0),(1,0),(0,1),(1,1), wherefirstcoordinaterepresentswhetherttheandwastrue, andsecondcoordinatewhethert$

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 sets (different weights resembling different functions and then different information extracted from the raw inputs)

this power of merging multiple simple units to overcome far complex problems it's the heart idea laying behind neural networks. We have seen in the previous example what it's known as a multilayer perceptron. Here, a perceptron resembles what it is known in deep learning as a neuron. The difference is, that perceptron computes a non differentiable function with the inputs received I(wx +b) whereas in neural networks, differentiable functions (activation) are used.

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, so that we don't have to specify functions(better another word?) (or, and, like we have seen), but let the model and the **optimization** process decide which are the best weights, for building better functions that transform the problem into something more easy to become linearly separable in each **layer**, until hopefully, the knot being completely untied at the end of the whole pass through the net. Many network layers == Sequence of simple nonlinear coordinate transformations.

And don't forget that "allowing the optimization" means that these weights are computed from the data (experience) provided to the system. So it's the data "human corroborated truth" which will allow the data for which we want to predict an outcome (unkown questions) to receive an answer. And the model being a means intermediary facilitating this process.

Thus we have...

"formal" definition of FFNN function, loss, update step, (backpropagation gradient descent) to measure how we're doing, instead of how many wrong classified, we have loss functions... different stop criteria e.g. after number of epochs, when ev loss starts growing training increases, after a certain number of updates

Perceptron equals FeedForwardNeuralNetwork It is structured data input of fixed variable, each component having its own meaning.

The amount of different models is gigantic There are other models suitable for unstructured data, such as sequences of text, which are the input to MT systems

The XOR problem, Perceptron, neural network concepts 1960,70,80 their stablishment is recent though. we can see 2 milestones that put them as de facto model 2011, nn in ASR Large vocabulary continuous speech recognition with context-dependent 2012 Alexnet CNN Imagenet, were each year the progress was only 2pcnt per year, supposed a huge climb Much more computational capacity with gpus that can perform matrix multiplications fast (required for backpropagation) internet and growth of laaaaaaaaaarge labeled datasets to feed enough data to large enough models so that they don't overfit

And das ist alles

1.4 Machine Translation

(mention somewhere about how to view mt as a classification task) 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 appeerence of intelligence, the intelligence required to find how a concept encapsulated in words of one language, can be expressed in words of another language. To a person this task often requires profound knowledge of both languages, unraveling meaning and arranging a good representation of this meaning in words from a complete different set(another language) 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 this encapsulating of concepts and in (distiling) disentangling them from their representations, even when validated by native speakers.

We can distinguish how historical attemps 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 in production and involving the work of experts in the languages to build these systems.

However, the predeominant and recent trend has been the use of statistical methods. As we've seen, trying to **learn** how to perform the task with the use of a model that is built up from data, from experience. This means having well translated sentences (big amounts of them) on the two languages that we're considering, and from which the model will (learn to) take decisions.

Predict output, answer questions, make a guess.

Formally the sentence to be translated is represented as an array of **n** words x = (x1,x2,x3,...,xn) and the translation of the sentence with the array of m words y = (y1,y2,y3,...,ym). These are the source and target sentences respectively.

As in the supervised classification scenario or pattern recognition task that we have showcased previously, our ideal goal is to result in the y maximizing the probability p(y | x), in this case, result in the best translation possible, the set of words y that is in most probability a translation of x. This would be the purest translation of x. The one that match maximally the concepts that x encloses.

y = argmax p(y | x)

as we've seen before, bayes rule can be used to transform the expression of this optimization and we have

y = argmax p(x | y)p(y) / p(x) = argmax p(x | y)p(y)

(ibm this with 2 parts) (nmt directly p(y | x)) we can focus on 2 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(y) and p(x | y) respectively.

Word based models, 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, 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 (1993 ref) was an example of a word based model. For computing an estimation to p(x | y), the translation model, it relied mainly on two aspects. First, a way to learn from ground truths (well (human) translated sentence pairs) what is a best translation for each word individually, 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 (ref), again, based too 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(y), the language model, n-gram models 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 to 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(y) would be to think of which is the probability of the next word in a sentence yn given the previous words, and then give the probability for the whole sentence using the chain rule. p(y) = p(yn | yn-1, yn-2, ..., y0) * p(yn-1 | yn-2, ... y0) * ... * p(y1 | y0) * p(y0) 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 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. (cursive) Although we will not dive into an explanation of these constituents, they were the state of the art before deep neural networks took main presence.

1.5 Neural Machine Translation

as we've seen in section (deep learning), the use of neural networks for classification tasks started to show results that created performance gaps in comparison to non deep systems. ffnn worked for structured data and cnn worked best for images. as these outcomes appeared each time more present, the 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 (reference to variable length ...) a RNN with an attention mecanhism.

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 (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) (which can also be seen as a classification task with an very wide clasification space)

A RNN is a neural network which incorporates an internal (cursive) hidden state. 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 x. This means, it will capture information about the preceding words (or future words of the sequence too in a BiRNN [**?**]) when working at word at position n.

p(y | x) = sum p(y,h | x)

these models and the later transformer achitecture, model p(y | x) (reference to eq) directly, in contrast to the non-neural approaches described in section (...). Even we denote n as the length of the sequence, a rnn expects inputs of variable n, while in a ffnn for example, once the model is defined, the input length remains constant.

Each state hi is built often as a lstm unit or a gru unit, that determines which information should be kept from the previous states and from the upcoming word, to be passed to the next (or precedent) state. If we combine these two RNN possibilities, one building h from the start of the sentence to the end and the other in inverse order, we would be describing a bidirectional RNN. (reference M Schuster and K. K. Paliwal. "Bidirectional recurrent neural networks) h = phi(Wx + Wh1 + b) h = phi(Wx + Wh1 + b) (phi being a nonlinear function which for example vary if we use lstms instead of gru) When we reach the final word of the sentence (t=T), h has been built upon preceding states (h) and all words have been read. Thus we can see 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. (idea that in this representation we don't have isolated words, but a vector where looking at each element has information of the other words, then it is richer)?

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 (...) and the task of discovering which is a proper set of weights 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 h as input, produce word per word, a sentence in the desired language of translation. The internal state of this decoding network updated with the preceding state together with the word the system just previously produced until the end of sentence is brought out. s=phi(s-1, y-1)

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 RNN nor NN. As mentioned in the beginning of this section, it was the model used in (reference to variable length ...), 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 nn (3 RNN joined together) and the whole trained together at a time.

It included though an element of main relevance, an (cursive) attention mechanism that modified the update step of the internal state in the decoder RNN to depended on an extra (cursive) context vector. s=phi(s-1, y-1, c)

Attention mechanism

Attention is a technique that (in a RNN) allows an state h to receive information directly from any other state hy (lookup into the past or into the future). If we see hi as the maximum 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, 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 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 (kevin murphy ref) attention can be described as a dictionary lookup where for a query q and a set of keys ki, we compute a combination of values vi A(q, (k, v)) =

sum alpha(q,ki)vi the strength of the connection between the query and the key i modeled by the weight 0<alpha(.)<1 alpha(q,ki) = softmax(a(q,ki)) were a can a function with trainable weights a = wTv tanh(Wqq + Wkk) In the context of a rnn with hidden states representing word positions, attention could compute the dependency of hidden state h_t to each other state hi vi,ki = hi, q = wstate at time t or word to be produced at time t A(xn, hi) = sum alpha(xn,hi)hi A(sn, hi) = sum alpha(sn,hi)hi Thus attention when focusing on a word (or representative of the word) 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 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's been shown 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.

The first system used attention during the decoding. As the decoder received the fixed representation of the whole sentence in the last state of the encoder h, it allowed to dinamically focus on parts of the source of sentence upon each production of a word for the target sentence.

As seen in eq s=phi ... c = ... 15.45 del KM

The attention weights can be seen acting as an alignment of target words to source words, hence the title of the article "NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE"

1.5.1. The Transformer Model

p(y|x) If the first neural standalone model was released in 2016, it wasn't long before the Transformer model was published in 2017 (before google published the Transformer model in 2017) If models using attention were showing state of the art results in the area, the transformer based it's 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 word result in the decoder.

Given the wide employment of the model, there are many good references explaining it[,,,], 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 real(istical) grasp on the tasks carried out along the project

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.

Multi Headed Attention When a sentence is supplied to the system, we don't have 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, and 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 xi and output word yi, yi = Attn(xi; (x1; x1); ... ; (xn;

xn)) In practice attention is carried out through matrix operations Attention(Q; K; V) = softmax(QK$^{T}/sqrt(dk))VWheretheproductisdividedbythescalingfactortopreventverysamllgradientsproblema$

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 (cursive) head could focus on adjectives that are describing it. (each one of these can also be computed in parallel) (The interpretation of the dependencies becomes blurred as we go deep in the network, although it's been shown necessary to achieve best results.) headi = Attention(QWiQ; KWiK; V WiV) Where queries, keys and values have been linearly projected to other subspaces by the trainable matrices Wiq,k,v Heads are concatenated, and to restore the original dimension, multiplied by Wo MultiHead(Q; K; V) = Concat(head1; ... ; headh)WO

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 (cursive) 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, theres a layer of multi head attention that attends source sentence information. Queries are the current representation for output words yi 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, ffnn with relu activations (ref to previous section) are used to build the units. After both, ffnn or attention, layer normalization(ref in attention all u need) and residual connections (ref)are performed. These are common techniques in nn that help the optimization process of the net. Layer normalization aims to prevent the problem of vanishing and exploding gradients (see KM chap 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.

Layer normalization computes the mean and variance across outpus of hidden units of a layer (a$_{i}^{l} forhiddenunitiooflayerl)beforetheactivationfunction.ai = wi * hlwherewiaretheweightsofthehidd$

a$^{i} = (aii)/ianda\ i = a^{i} + (KM)$

y and b being trainable weights that allow the net to undo these changes partially or completely favouring the optimization

Residual connections allow the input of a layer to (cursive) skip all the transformations F that the layer perform to x, thus add x at the end of the layer. F′(x) = F(x) + x. F′(x) being the output of the residual block. This is why they are also called skip connections.

Gathering the elements described, we have the building blocks of the architecture. def EncoderBlock(X): Z = LayerNorm(MultiHeadAttn(Q=X, K=X, V=X) + X) E = LayerNorm(FeedForward(Z) + Z) return E

def DecoderBlock(X, E): Z = LayerNorm(MultiHeadAttn(Q=X, K=X, V=X) + X) Z′ = LayerNorm(MultiHeadAttn(Q=Z, K=E, V=E) + Z) D = LayerNorm(FeedForward(Z′) + Z′) return D

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 (textbf? cursive?) positional embedding is combined with the (cursive) 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 X vocabSize, where n is the number of words.

After this, it is multiplied by a learnt embedding matrice Ee, of size vocabSize x d, where d is a parameter of the model, the dimension to which words are projected, ending with a representation of a sentence in dimension n x d. This is the (cursive) word embedding.

WE = x$_o ne_h ot * Ee$

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

PE(pos;2i) = sin(pos=100002i=dmodel) PE(pos;2i+1) = cos(pos=100002i=dmodel)

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

This design choice provided no performance decrement, while allowed the model to extrapolate to sentence lenghts unseen during training.

Exactly the same process is applied to source words on the encoding as to words that enter the decoder stack(yt-1 forall t produced by the decoder in previous instants).

The model uses a technique called Weight Tying (ref) where a single matrix E for the encoding as for the decoding is used, and also for the output embedding matrix(E^t)$, thattransformstheoutpu$

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, BERT, or T5 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. Recent results of the 2022 IWSLT competition (https://aclanthology.org/2022.iwslt-1.10.pdf) show the wide and present use of the Transformer model for machine translation tasks.

—Modelo usado en tfg –descripcion, ventajas, transformer, resultados, citar competiciones, trabajos, articulo que aglutina resultados de la competicion IWSLT 2022, gran

Table 1.1: 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
v3FTk90	29.5	35.3	40.2	45.5	**41.6**	**42.9**
v3FTk100	29.5	35.3	40.2	45.5	**42.0**	**43.1**

mayoria transformer –Dejar clara que la decision se basa en resultados recientes –mencionar que es el utilizado

Evaluation

problematic of evaluations bleu article reference ter

1.6 Framework of this work

1.7 Document structure

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

CHAPTER 2
Data

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

2.1 Preprocessing

2.1.1. Tokenize

2.1.2. Truecase

2.1.3. Byte Pair Encoding

2.2 Filtering

2.2.1. Sentence length ratio

2.2.2. Language ID

2.3 Datasets

2.3.1. Europarl-ST

2.3.2. CERN

2.3.3. WMT

2.3.4. Polimedia

2.3.5. CERN-News

CHAPTER 3
Europarl-ST

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

3.1 ?? ???? ???? ? ?? ??

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

CHAPTER 4

Conclusions

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

Bibliography

[1] Jennifer S. Light. When computers were women. *Technology and Culture*, 40:3:455–483, juliol, 1999.

[2] Georges Ifrah. *Historia universal de las cifras*. Espasa Calpe, S.A., Madrid, sisena edició, 2008.

[3] Comunicat de premsa del Departament de la Guerra, emés el 16 de febrer de 1946. Consultat a `http://americanhistory.si.edu/comphist/pr1.pdf`.

APPENDIX A

Configuració del sistema

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

A.1 Fase d'inicialització

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

A.2 Identificació de dispositius

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

APPENDIX B

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

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