Master Thesis

Evaluating the Performance of Neural Machine Translation in Federated Learning System

Diego Watanabe

Master Thesis DKE

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science of Artificial Intelligence
at the Department of Data Science and Knowledge Engineering
of the Maastricht University

Thesis Committee:

Dr. Jan Niehues
Dr. Anna Wilbik

Maastricht University
Faculty of Science and Engineering
Department of Data Science and Knowledge Engineering

June 28, 2022

Abstract

Privacy has been in the media attention since a few years ago. One of the biggest cases was the Facebook-Cambridge Analytical data scandal where the company was using physiological profiles of users to influence political decisions [6]. Although this issue is one of the awareness of many people, some solutions were developed to tackle the problem. One of them is Federated Learning created by Google in 2017. In a traditional centralized system, all data is sent to the model located in the server. The idea of Federated Learning is to run the machine learning model in a decentralized system to keep the users' data on their own devices. In this study, we focus on text translations using Neural Machine Translation based on the Federated Learning approach. We run six different systems. Each of them follows a strategy on Federated Learning techniques focused on parallelization, or sequential training. For the experiments, we use a parallel dataset and different models to compare the performance of the systems. The results of the experiments showed that the loss in the final model is higher in systems that have to average many client models. Depending on the FL system strategy, the improvement in BLEU results could be between 1 and 4 for 5 and 20 clients. Finally, we remark on the impact of the dataset on the systems' performance.

Contents

Chapter 1

Introduction

For thousands of years, humans have conducted business across geographic, political, and cultural boundaries. To arrange the communication between different cultures, a language is needed. Even in our days, when English is widely extended and accepted as the most popular spoken language worldwide, there are some situations in that we need to translate texts.

Due to the fact that human language is ambiguous and flexible, translations have been a big challenge for the science community since it appears in 1933 when George Artsrouni designed a storage device on paper tape. The device could be used to find the equivalent of any word in another language [5]. Neural Machine Translation (section 2.1) is a technique of Machine Translation that, using Neural Networks, translates the text from one language to another. This process is made by an encoder, which reads the input to produce a representation and a decoder that generates the translation.

Commonly, the machine learning model is hosted in a server and the data is sent to it to be processed. These generate security and privacy problems if the data contains personal information about users. In Federated Learning, the server sends the model to the devices, and the training process has done in each device, also called client. By this, the dataset of each user never leaves the device, and it can be more protected against attacks. An example of this technique is Gboard in Android [4]. The phone stores information about the current context and whether the user clicked the suggestion. Following, the keyboard shows a suggested query with the personalized information of each user.

In this study, we use NMT in a Federated Learning system to evaluate the performance of the translations using different methods to update the model like Federated Averaging (FedAvg) and Federated Stochastic Gradient Descent (FedSGD). The evaluations are tested by different experiments using a particulars number of models and epochs, and two models: CNN and Transformers. To measure the results, we use Bilingual Evaluation Understanding (BLEU4), which is an algorithm to evaluate the quality of the translations. It is explained in more detail further, as well as the other definitions. Due to training NMT models in FL settings has been barely explored, in this study we try to focus on the efficent

of the FL system and the parameters that influence on the final performance. To do that, we answer the next Research Questions:

1. How does the number of clients influence the performance of a Federated Learning?

2. What is the impact of the size of the training set on performance? More specifically, how much data is needed for efficient translation?

The two datasets selected for the experiments are Europarl and TED. Each of them has a specific Spanish-English and Dutch-English parallel dataset, both of them are explained in Chapter 3. The data collection of this study is based on the supervisors' suggestions and the comparisons with other datasets available in the network. The thesis structure is the next: Chapter 2 is based on the evolution of Machine Translations and the tools and techniques used during the thesis. Chapter 3 describes the two datasets used, explaining the type and size. Chapter 4 is about what the thesis is about, their architecture, and in the next chapter, the experiments. The last two chapters are related to the conclusion sets after analyzing the experiments and the future work to continue with this research.

Chapter 2

Background & Related Work

In this section, we explain the basic concepts and terminology to understand the purpose of this thesis. First, we explain what is Machine Translation and its evolution until now. Then, we go through Federated Learning and describe the different architectures that we use during this study. To finalize, we explain how we measure the quality of the results using Bilingual Evaluation Understanding (BLEU4). Before starting with the description of the mechanisms, we explain the evolution of Machine Translation.

Communication has been a problem since civilizations started interacting with each other. The first machine translation was introduced in the Soviet Union by Peter Troyanskii where they took the first word from the text, found a corresponding card, took a photo, and typed its morphological characteristics. Since that time the evolution of Machine Translation has passed through different steps:

- *Rule-based machine translation (RBMT)*: It is based on linguistic information that covers the main semantic morphological, and syntactic regularities of each language. [19]
- *Statistical Machine Translation (SMT)*: Unlike RBMT, the translations are generated based on statistical models. It analyzed similar texts in two languages and tried to understand the patterns. The analysis can be based on words, phrases, or syntax. It is a corpus-based machine translation because is generated on the analysis of bilingual text corpora. [10]
- *Neural Machine Translation(NMT)*: It is an approach that uses a neural network to predict the probability of a sequence of words based on the encoding and decoding of the text. It is also a corpus-based machine translation like SMT. [9]

2.1 Neural Machine Translation

Neural Machine Translation (NMT) uses neural networks to translate from the source text to the target text. It used the idea of codifying the words into a series of numbers, vectors, or matrix representations. The core of these works lies an encoder-decoder architecture. The encoder processes a variable-length input (source sentence) and builds a fixed-length vector representation. Conditioned on the encoded representation, the decoder generates a variable-length sequence (target sentence) [2]. The key to the encoder-decoder architecture is the ability of the model to encode the source text into an internal fixed-length representation called the context vector. It could happen that, once encoded, different decoding systems could be used, in principle, to translate the context into different languages.

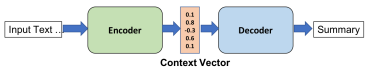

Figure 2.1: Encoder-Decoder architecture

2.1.1 Encoder-Decoder Model

These models are limited by a fixed-length input sequence where the output must be the same length. The workflow is explained in the next paragraph.

An encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder-decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence[1].

2.1.2 Encoder-Decoder with Attention

The Encoder-Decoder architecture has problems with long sequences of text to be translated. The problem stems from the fixed-length internal representation that must be used to decode each word in the output sequence.

The solution is the use of an attention mechanism that allows the model to learn where to place attention on the input sequence as each word of the output sequence is decoded.

Attention layers are explained in more detail in section 2.2 Attention layer.

2.1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are mostly used in image recognition and image processing. However, in 2017, Facebook presented Convolutional Sequence to Sequence Learning [3]. Contrary to recurrent models, convolutional models'

computation can be fully parallelized to better exploit the GPU hardware. This makes CNN faster than Recurrent Neural Networks (RNN). [14]

First, input elements are embedded in distributional space. Also, the model adds a sense of order by embedding the absolute position of input elements. Both are combined to obtain input element representations. The same concept is applied for output elements that were already generated by the decoder network to obtain output element representations.

Both encoder and decoder networks share a simple block structure that computes intermediate states based on a fixed number of input elements. Each block contains a one-dimensional convolution followed by a non-linearity. Stacking several blocks on top of each other increases the number of input elements represented in a state. Gated linear units citedauphin2017language was chosen as non-linearity which implements a simple gating mechanism over the output of the convolution.

To enable deep convolutional networks, residual connections were added from the input of each convolution to the output of the block. For encoder networks, the output of the convolutional layers matches the input length by padding the input at each layer. However, for decoder networks, we have to take care that no future information is available to the decoder [21].

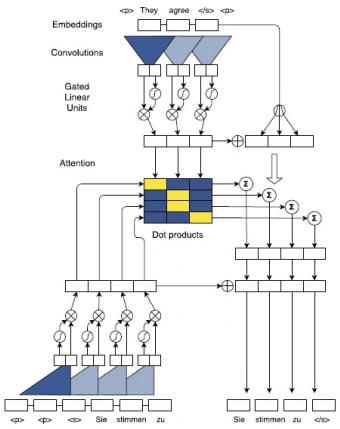

Figure 2.2: Change name

Each convolution kernel is parameterized and takes input which is a concatenation of inputs elements embedded in dimensions and maps them to a single output element that has twice the dimensionality of the input elements; subsequent layers operate over the output elements of the previous layer.

To enable deep convolutional networks, the models adds residual connections from the input of each convolution to the output of the block.

For encoder networks, the output of the convolutional layers has to match the input length by padding the input at each layer. However, for decoder networks, no future information is available to the decoder [21].

Finally, there is a computation of a distribution over the possible next target elements by transforming the top decoder output via a linear layer with weights and bias.

2.2 Attention layer

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence to compute a representation of the sequence.

Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution.

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

When the keys, values, and queries are generated from the same sequence it is called self-attention. If the attention mechanism is between the encoder and the decoder, it is called cross-attention since keys and values are generated by a different sequence than queries. Self attention is illustrated in purple and cross attention in red in figure 2.3.

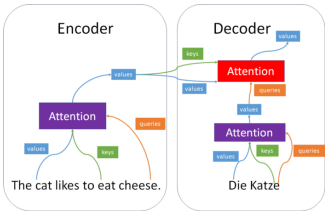

Figure 2.3: Change name

Instead of performing a single attention function, the operations are in parallel. The multi-head self-attention extracts a weighted key (K), value (V), and query (Q) from each input (X), and calculates an output matrix for each head as follows:

$$Head = Softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

$$withQ = XW_q, k = XW_q, andV = XW_v \tag{2.2}$$

where dk is the dimension of the queries and keys. Then, the outputs of all the heads are concatenated and weighted again. It can be formulated as:

$$MultiHead = Concat[Head_1, ..., Head_n]W_o \tag{2.3}$$

2.2.1 Transformers

Transformers appeared for the first time in 2017. They are a type of machine learning model based on the Attention mechanism [22]. Since there, Transformers are not only used in NLP but is expanding to other areas like Computer Vision [7] or Reinforcement Learning [17].

The goal of Transformers is to take a sequence and outputs it in another target language. It consists of two blocks: encoder and decoder, it is explained in Encoder-Decoder with Attention. After tokenizing the text into tokens, each of them is converted to its id. This process is embedding, and it only happens in the first encoder. After receiving a list of vectors as input, an encoder processes this list by passing these vectors into a self-attention layer that helps to look at other words of the sentences' id, then into a feed-forward neural network, then sends out the output upwards to the next encoder. There are dependencies between the ids in the self-attention. The output of the self-attention layer goes to a feed-forward neural network for training and then sends the output to the next encoder. The last encoder's output transforms its output into a set of attention vectors K and V. These vectors are used by the encoder-decoder attention layer of each decoder which helps it to focus on appropriate places in the input sequence.

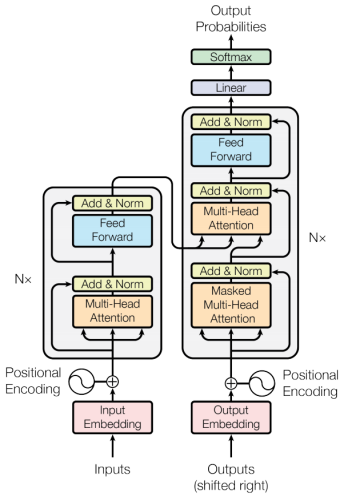

Figure 2.4: Transformer architecture

The decoder is similar to the encoder, but it contains an attention layer between the self-attention and the Feed Forward layer. Like the encoder, the output of the first decoder feeds the next decoder and repeats the same behavior until the last one. The last decoder's output is transformed into a logits vector by a Linear layer. This logits vector represents the words of the target language, the Softmax layer converts into probabilities and the cell with the highest one is chosen.

This mechanism makes the Transformers training phase significantly faster than using recurrent or convolutional layers.

2.3 Byte-pair encoding

NMT models typically operate with a fixed vocabulary, but translation is an open-vocabulary problem. This makes NMT models are affected by the Out of Vocabulary (OOV) and rare word problems, thus it degrades the translation quality. OOV words are the words that are not appearing in the corpus and rare words are the words that appear very few times in the corpus. When translating unknown words, they are replaced with UNK tokens. Therefore, translations become worse since these meaningless tokens increase the ambiguity by breaking the sentence structure.

Character segmentation is a technique used in machine translation to avoid the drawbacks of word-level translation. The advantage of character segmentation is that it can model any compositions of characters, enabling better modeling of rare morphological variants. However, the improvements may not be much significant due to missing important information since the character level is more fine-grained.

To face these issues, Sennrich [18] introduced the concept of segmenting words into sequences of subword units by providing a more meaningful representation. As an example for subword segmentation, consider the word "looked". This word can be split into "look" and "ed". In other words, two vectors are used to represent "looked". Therefore, even if this word is unknown, still the model can translate the word accurately by treating it as a sequence of subword units.

BPE is used for word segmentation to allow the representation of an open vocabulary through a fixed-size vocabulary [18]. The technique was adapted to merge characters or character sequences, instead of frequent pairs of bytes. As a result, the NMT is capable to represent rare and unseen words as a sequence of subword units. In this study, the models use a BPE vocabulary. So the text has to be encoded before the translation. That is why previous to BPE, input text needs to be tokenized. BPE ensures that the most common words are represented in the vocabulary as a single token while the rare words are broken down into two or more subword tokens and this is in agreement with what a subword-based tokenization algorithm does. To represent the corpus most efficiently, BPE goes through every word to merge at each iteration by looking at its tokens frequency. Resulting in a good performance, making BPE one of the most widely used subword-tokenization algorithms.

2.4 Federated Learning

In a standard Machine Learning system, the model and the data are in one machine. This means that communication costs are relatively small, and computational costs dominate, with much of the recent emphasis being on using

GPUs to lower these costs. This approach brings a problem when we work with sensitive data, for instance: users' private information. In 2017, Google proposed a new approach to solve this issue, Federated Learning (FL) [12]. The idea of FL is to send the model to each user and keep their data on their own devices. Each client trains the model and sends the updates to the server (Figure 2.5). It averaged and build a new model. However, this approach is limited by an upload bandwidth of 1 MB/s or less. Further, clients will typically only volunteer to participate in the optimization when they are charged, plugged in, and on an unmetered wi-fi connection. Further, we expect each client will only participate in a small number of update rounds per day.

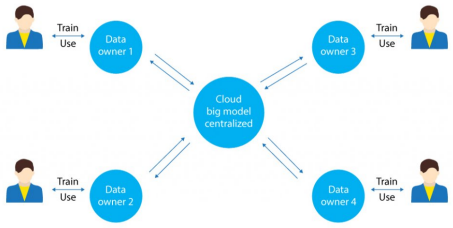

Figure 2.5: Federated Learning

Federated Learning (FL) is a decentralized machine learning technique that focuses on clients' data protection. To do this, FL keeps the data on their own devices. Also, it can work in a more complicated environment than distributed systems because it works in an unbalanced number of clients that can be disconnected from the server. For instance, mobile phones without internet connection. The numbers of clients in FL depend on the environment, more specifically, on the privacy of the data. If the data is not shared, the number of clients is fixed. There are two primary ways we can add computation: 1) increased parallelism, where we use more clients working independently between each communication round; and, 2) increased computation on each client, where rather than performing a simple computation like a gradient calculation, each client performs a more complex calculation between each communication round.

One of the most relevant parts of FL is the aggregation method. After the client train the model locally, it sends the parameters to the server to apply an aggregation method to build a system machine learning model. There are many aggregation methods to calculate the final weights of the model. In this study, we focus on two of the most common: Federated averaging and Federated stochastic gradient descent. Before start explaining the two methods, we define the parameters that are essential to understand the algorithms used in the aggregation methods.

- The fraction of clients on each round that computes the gradient of the loss is defined by the parameter C.
- E as the number of training passes each client makes over its local dataset on each round

- B is the local minibatch size used for the client updates.
- The learning rate is η.
- K is the client that computes $G_k = \bigtriangledown F_k(w_t)$.
- Current model W.

2.4.1 Federated Stochastic Gradient Descent

Federated Stochastic Gradient Descent (FedSGD), as its name said, is based on Stochastic Gradient Descent (SGD) optimization. It works with a part of the total clients that perform computation in each round and treat the local data as a single minibatch. To apply this approach in the federated setting, we select a C fraction of clients on each round and compute the gradient of the loss over all the data held by these clients. Thus, C controls the global batch size, with $C = 1$ corresponding to full-batch (non-stochastic) gradient descent [13]. The central server aggregates gradients $G_k = \bigtriangledown F_k(w_t)$ of each client and applies the update $w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k$. An equivalent update is given to each client $\forall k, w_{t+1}^k \leftarrow w_t - \eta g_k$ and then $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$. That is, each client locally takes one step of gradient descent on the current model using its local data, and the server then takes a weighted average of the resulting models. The amount of computation is controlled by parameters C, E, and B.

2.4.2 Federated Average

Federated Average (FedAvg) is similar to FedSGD, but instead of working with some clients per round, it works with the global amount of clients in the system. In FedAvg, like FedSGD, clients keep their data locally for privacy protection; a central parameter server is used to communicate between clients. This central server distributes the parameters to each client and collects the updated parameters from clients. In FedSGD, each client locally takes one step of gradient descent on the current model using its local data. It is possible to add more computation to each client by iterating the local update $w^k \leftarrow w^k - \eta \bigtriangledown F_k(w^k)$ multiple times before the averaging step. FedAvg is mostly studied in centralized fashions, which requires massive communication between server and clients in each communication [20].

2.5 Transfer Learning

In the experiments that we run during this study, the models have to deal with a huge dataset (Europarl), making the training phase takes a very long time. Instead of training the model every time, we need a technique to train once and use the already trained model to retrain it again with the other dataset (TED).

Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. According to Lisa Torrey and Jude Shavlik [15], TL offers three possible benefits to look for when using it:

- *Higher start*: The initial skill (before refining the model) on the source model is higher than it otherwise would be.
- *Higher slope*: The rate of improvement of skill during training of the source model is steeper than it otherwise would be.
- *Higher asymptote*: The converged skill of the trained model is better than it otherwise would be.

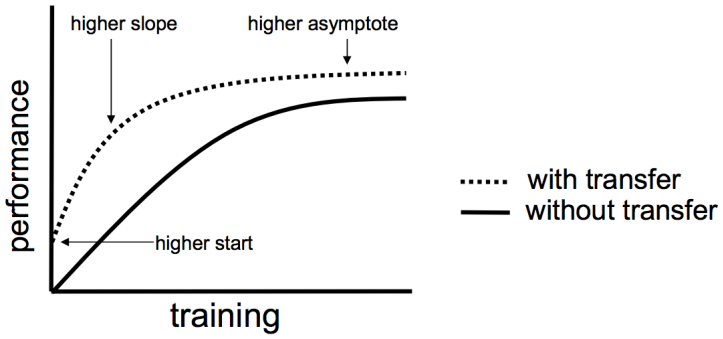

Figure 2.6: Transfer LEarning

Due to neural methods are data-hungry and learning poorly from low-count events, TL is a good solution to bring good performance [23].

The TL approach we use is simple and effective. We first train an NMT model on a dataset where there is a large amount of bilingual data (Europarl dataset), which we call the parent model. This new model is then trained on a dataset with little bilingual data (TED dataset), which we call the child model. This means that the low-data NMT model will not start with random weights, but with the weights from the parent model.

2.6 Evaluation Metric

Translation of text are not a classification task, so we have to use an algorithm to evaluate the quality of the translations. This quality is how similar is the translation of a machine and a human. The more similar, the best. In most cases, it is possible to translate a phrase into more than one sentence and all of them can be "perfect" translations. This makes the task ambiguous, which is difficult for humans, and even more difficult for machines.

In this paper, we use Bilingual Evaluation Understudy (BLEU). It is an algorithm used to generate a score describing the quality of a corpus translation in a range between 0 and 1. Score the maximum is almost impossible. Even human translations do not reach that value.

Normally, MT traductions contain more than one word (Figure 2.7). To measure the quality of translation is by counting the times that the words appear in the references (Precision). As we can see, it is not accurate because we can obtain a maximum score on a bad translation (7/7=1). BLEU modified the precision by only counting the maximum number of times that appear in the reference sentences [13]. In this case, reference 1 has a score of 2/7. Also, it adds a brevity penalty value to penalize short translations.

Candidate: the the the the the the the.
Reference 1: The cat is on the mat.
Reference 2: There is a cat on the mat.

Figure 2.7: Precision example

It also works on bigrams, which means that it evaluates more than one word of the sentence. In the example of figure 2.7, we have the next subsentences:

	count	Count Clip
the cat	2	1
cat the	1	0
cat on	1	1
on the	1	1
the mat	1	1

In this case, we calculate the number of times the subsentences appear in reference one and reference two (count). And the number of times the subsentences appear in reference one or reference two (Count Clip). Finally, the two values are divided to obtain the score (4/6).

It can focus on one word, which is unigram, or n-gram. The equation to calculate it is the next one:

$$BP = \begin{cases} 1 & if MT\_olength > ref\_olenght \\ \exp(1 - MT\_olength/ref\_olength) & otherwise \end{cases}$$

2.7 FAIRSEQ

FAIRSEQ is an open-source sequence modeling toolkit based on Pytorch to execute different tasks like train custom models for translation, summarization, language modeling, etc. FAIRSEQ is different from other toolkits for its common interface across models and tasks that can be extended, and pretrained models for machine translation, summarization, and language modeling. [16]

In this study, we focus on the Machine Translation implementations of sequence-to-sequence models. We use the convolutional model and a Transformer for the FL tasks. For that goal, we use dictionaries to adapt the dataset of the two models. In addition, we use the full state of the model (checkpoint), so that

results are reproducible if training is interrupted and resumed. When an epoch finishes in the training step, FAIRSEQ generates a checkpoint of the number of the epoch and the best checkpoint so far. This best checkpoint is the one with the lowest validation loss from all the checkpoints.

Chapter 3

Dataset

A dataset is necessary to train and evaluate a Machine Translation. It has to be a parallel dataset that contains two files: one with phrases of the source language and the second one with the same sentences in the target language that express the same meaning in different languages. Both two files of the dataset are sentence alignment, which means that express the same meaning in different languages [11]. In our case, we work with two datasets: Europarl and TED. We select these two datasets due to the size of each of them. Europarl is almost seven times bigger than TED. These differences bring us a chance to examine the impact of the size of data on the results. The language to focus on is Spanish to English and Dutch to English. The number of characters in each passage in parallel passages is limited to not bigger than 700, to make the passage not too long for the models to process it. Focusing on the content of the datasets, TED includes more topics than Europarl. While Europarl speeches are related to political decisions, TED has a variety of topics and, in most cases, they are informal dialogues. In table 3.2, we can see the difference between the two datasets, and in table 3.1, the size of them for Spanish and Dutch.

3.1 Europarl

The European Parliament is the forum of the European Union where parliamentarians discuss and make decisions at the European level. The members of the European Parliament are elected by voters in all Member States to represent people's interests in EU law-making. To reach this goal, the members exposed their ideas in debates. Due to the members speak in different languages, it is necessary to translate them to their mother tongue.

The Europarl dataset contains the speeches of the parliamentarians in the Europearl Parliament for many years. It consists of 21 European languages. Initially, it was created for statistical machine translation systems but, in this study, we use it for training the NMT. All the files are without tokenization [8].

3.2 TED

Technology, Entertainment, Design (TED) is a nonprofit organization that spreads ideas through powerful talks (19 minutes or less). They covered almost every topic – science, business, global issues, etc – in more than 100 languages. TED community consists of people from every discipline and culture with a seek of deep understanding of the world.

TED dataset contains nearly 4000 TED conferences translated by a team of volunteers to more than 100 languages.

	Spanish	Dutch
Europarl	1.960K	1.978K
TED	405K	312K
Total	2.3M	2.1M

Table 3.1: Number of sentences in the datasets.

Europarl.	TED
So, judicially, something needs to be done.	It was a fairy tale, and then we retired.
I ask the Commission what can be done to speed up implementation in this particular area.	Then she took me to see a film that I really didn't want to see.
In relation to legal certainty, I endorse the point made by Mrs Thyssen.	It ruined my life – (Laughter) "The Inconvenient Truth" and Mr. Gore.
It is important that business has legal certainty.	I have four kids, and even if part of what he says is true, they're not going to have the life that I had.
I mentioned this again to Commissioner Monti recently.	And I decided at that moment that I would spend the rest of my life doing whatever I could to improve their possibilities.

Table 3.2: Sentences from Europarl and TED datasets.

Chapter 4

NMT in a FL system

Due to NMTs being data-hungry, machine learning models need a large dataset to obtain satisfactory results. One solution to face this issue is using FL systems integrated by NMT. These models can be improved from the corpus held by different contributors without directly exposing them to one another.

After clarifying the sakes behind datasets and the models, we can explain the FL systems adapted to study their results. Like the neural network models' selection, we choose different systems. The criteria are based on complexity and parallel training.

4.1 NMT

The two models used in this study are CNN and Transformers, using the Europarl and TED datasets. Each of the datasets has Spanish and Dutch texts traductions to English. With this, we ensure that we can analyze the influence of the language on the performance, as well as the type of text. While Euoraparl has a purely political thematic, on the other hand, TED includes an extensive kind of topic. That makes TED talks more informal than Europarl, as we can see in table 3.2. Moreover, Europarl is seven times bigger than TED, so we can conclude that we consider size and quality to measure the performance.

For this reason, we assume that we can initialize the model with large training data, Europarl. Accordingly, the short dataset (TED) is used to evaluate the FL systems. That means that the low-data NMT model will not start with random weights, but with the weights from the previous model. Thus, the training time is reduced due to models are trained with Europarl once. The model must correspond with the vocabulary size to start training. That is why we pre-process the data using the Europarl dictionary.

With this, we investigate the influence of the FL system architecture in the NMT models (CNN and Transformers). Using different FL systems to determine the causes of their performances and the reasons behind each of them is the priority of the experiments.

4.2 Preprocessing

The goal of the preprocessing phase is to preprocess and binarize the dataset. The first step is to tokenize the text using a Perl script to split the sentences into words. Create the train, valid, and test files to apply BPE to obtain the best representation of text using the least number of tokens. In this phase, the dictionaries for the source and target language are created. This is not the case in the study because the dictionaries used during the experiments are the dictionaries from the first dataset, Europarl. With this, the size of the train, valid and test files, fit with the checkpoint used during the training and generation.

4.3 Training

The training phase is done in every client of the system using the baseline model at the beginning. Depending on the experiments, the model could be different (averaged model, baseline, or the model from the previous model). Also, it can be a CNN or Transformer, depending on the experiment that we focused on.

4.4 Generation

Finally, the generation of the results using the final model is the last part of the process. We take into account the value of the BLEU4 to compare them with other models. This phase is only done at the end of the experiments when the final model is available. We do not evaluate the model of every single client due to we assumed a better result, and it is not part of the goal of this study.

4.5 FL systems

The decentralized systems used in this study are focused on FL. Six strategies were used to train models and compared with the result of the baseline. The systems change on the number of clients that work in parallel, the merging of the models' weights, and the order of the training. The cases are explained in the next sections.

4.5.1 Sequential training.

The first case that we analyze is the basic one. The system starts training one client until it reaches the last epoch, sending the best checkpoint to the next client, and continuing the process until reaching the last client. As we can see, this process is sequential and without parallelization, which means that we are facing a strategy based purely on TL. The main disadvantage of this system is that the first models have poor performance. However, as the model training goes by, clients' models return better results due to they start training with the best checkpoint from the last client.

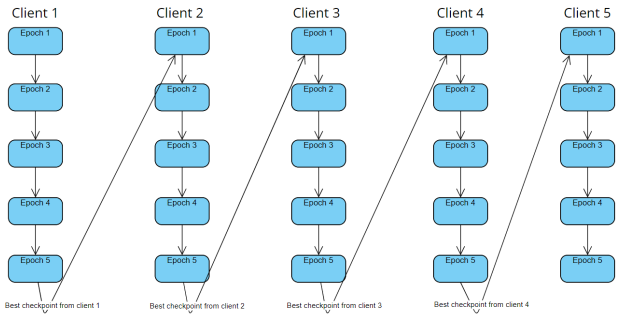

Figure 4.1: Sequential training

4.5.2 One epoch at a time

In the previous strategy, every epoch of the same client runs before the second client starts the training phase. However, in this case, the system only runs one epoch for a client and then sends the checkpoint to the next client. When the first epoch of the last client finishes, the model is used for the first client during the second epoch, and the workflow starts again. With this strategy, we try to remark the importance of reusing the checkpoint after a single epoch, preventing catastrophic forgetting.

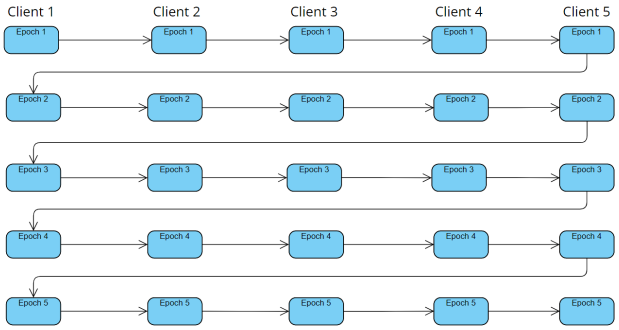

Figure 4.2: One epoch at a time

4.5.3 Five clients in parallel

In this case, we use parallelization to train more than one client at the same time. The system counts with n total number of clients where only k work in parallel. These k clients train with the same number of epochs and then average the weights of the models. The result is sent to the next k clients to repeat the process. Hereby, if n is equal to k, the system works with the FedAVG algorithm. In case n is greater than k, the algorithm used is FedSGD. During this study,

different values for k were tested. After evaluating them, the most suitable value that better fits is 5 in a system with n equal to 20.

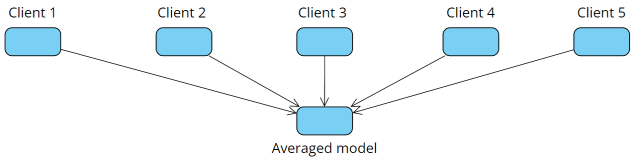

Figure 4.3: Five clients in parallel

4.5.4 Sequential training with two clients

For the fourth case, we based the system on FedSGD. The first k clients train for one epoch and average the resulting checkpoints to use in the next iteration. When the last epoch finishes, the averaged model is sent to the next k clients to repeat the process. If the total number of clients is odd, the last client will be processed alone. This strategy is similar to the first one, but instead of working with only one client, two clients work in parallel. With this, the training time is faster in most cases. The value of k is two to keep the parallelism in the system during the experiments. A greater even number like four would return a performance similar to the five clients in parallel which is not the goal.

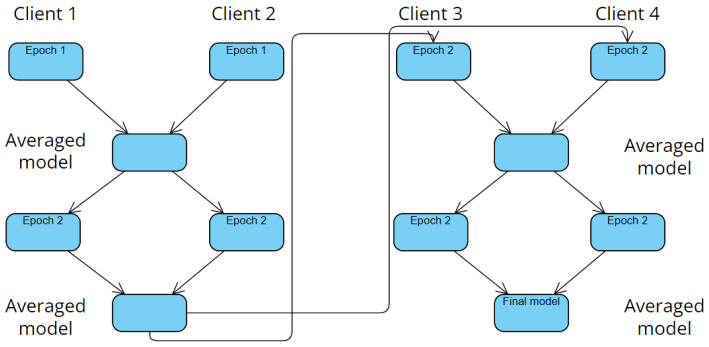

Figure 4.4: Sequential training with two clients

4.5.5 One epoch at a time with two clients

In the last strategy, the first k clients for epoch one are trained, average the checkpoints, and send the resulting model to the next k clients. These processes repeat until the system reaches the last k clients. The averaged model of the last k clients is used for the first k clients to start the training in the second

epoch and continue the process as before. This strategy is similar that case four. Instead of train and averaging the first k clients until the last epoch, it only runs once epoch and then sends the resulting averaged model to the next group of clients. It is relevant to mention that cases 4 and 5 are the same when the number of clients is equal and they are trained with a single epoch. During the study, the value of k is 2 to keep the symmetry with the last strategy.

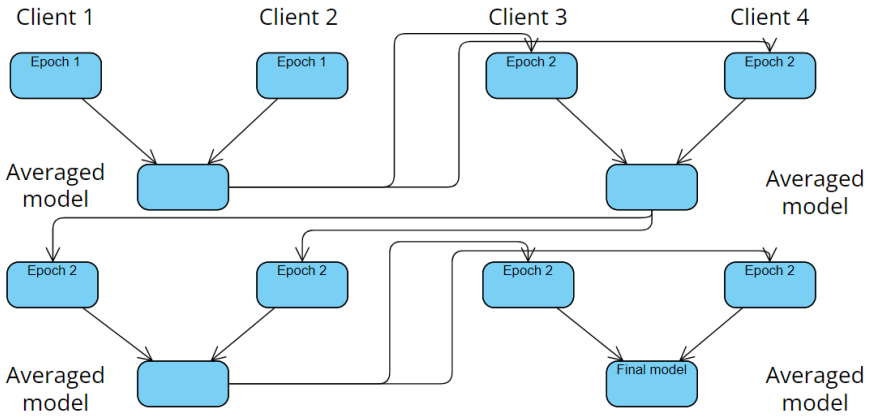

Figure 4.5: One epoch at a time with two clients

4.5.6 FedAVG

The last case is based on training all the clients in parallel, averaging the models, and training them again for the next epoch. This strategy is similar to case 3 if the total number of clients is five. The difference is that, in this case, the merging algorithm used in the system is FedAVG.

4.6 Baseline

Our baseline is a model trained with only one dataset, in this case, Europarl. It can be a CNN or a Transformer, but, in both cases, the model is trained with only one epoch.

After trying different configurations for the Transformer model on a low scale, the best architecture is the one that consists of two encoders and two decoders layers. We decide to keep the same number of layers for the CNN model. As expected, both model performance fails to successfully translate a different dataset from the one that it was trained. The results can be compared in table 4.1.

	Spanish	Dutch
CNN	14.39	12.98
Transformer	15.11	13.14

Table 4.1: Baseline results for CNN and Transformer.

Chapter 5

Experiments

In this chapter, we discuss the experiments that we run during the study. We mention the different environments used and explain the reason for the values of the variables.

The experiments were run in Google Colab using a single GPU. We focus on two models: CNN and Transformers. Both of them operate on an FL system as we explain in Section 4.

The goal of the experiment is to obtain results of the performance of the system using a different amount of clients and number of epochs, as well as different strategies to average the weights or run the models in parallel. To do this, the dataset is split into the same amount of clients in the system. For instance, if we have five clients, the dataset splits into five pieces. We determine a different number of epochs, which are crossover with the numbers of clients as we see in Table 5.1. In a real environment, it is not possible to determine the number of clients in a system because it depends on the devices available at that moment. For this reason, we evaluate different scenarios using a numbers of clients on the systems.

	5 clients	10 clients	20 clients	50 clients	70 clients
1 epoch					
5 epoch					
10 epoch					

Table 5.1: Example table

The first experiments run were focused on the CNN model and the ES-EN dataset. The number of clients in the system is 5, 10, 20, 50, and 70. All values are multiples of 5 to obtain a variety in the FL systems. On the other hand, the number of epochs during the tests are 1, 5, and 10. We do not consider values greater than ten because the models were overfitted in most systems. The

results of the experiments are from Tables 5.2 to 5.7 where the best system's result is highlighted.

	5 clients	10 clients	20 clients	50 clients	70 clients
1 epoch	**21.17**	20.46	19.39	17.88	16.51
5 epoch	**23.16**	21.9	20.87	19.35	17.61
10 epoch	**23.99**	22.82	21.11	19.06	17.67

Table 5.2: Sequential training

	5 clients	10 clients	20 clients	50 clients	70 clients
1 epoch	**21.17**	20.46	19.39	17.88	16.51
5 epoch	**24.09**	23.41	22.98	21.83	21.33
10 epoch	**25.28**	24.88	23.50	22.09	21.54

Table 5.3: One epoch at a time

	5 clients	10 clients	20 clients	50 clients	70 clients
1 epoch	**19.92**	19.72	19.53	19.27	18.80
5 epoch	22.01	**22.11**	21.97	21.28	20.89
10 epoch	**23.32**	22.71	22.39	21.38	20.83

Table 5.4: Five clients in parallel

	5 clients	10 clients	20 clients	50 clients	70 clients
1 epoch	20.04	20.52	**20.55**	19.93	19.55
5 epoch	**22.92**	22.54	22.35	20.84	20.42
10 epoch	**23.97**	23.63	22.78	21.95	21.31

Table 5.5: Sequential training with two clients

	5 clients	10 clients	20 clients	50 clients	70 clients
1 epoch	20.04	20.52	**20.55**	19.93	19.55
5 epoch	**23.10**	22.60	22.97	21.21	19.97
10 epoch	**24.60**	24.06	23.79	21.58	21

Table 5.6: One epoch at a time with two clients

	5 clients	10 clients	20 clients	50 clients	70 clients
1 epoch	**19.92**	18.52	17.11	15.49	14.29
5 epoch	**22.21**	21.16	19.81	18.31	17
10 epoch	**23.32**	22.06	20.57	18.17	17.32

Table 5.7: FedAVG

As we can see, the performance of the systems with 20 clients or more is almost equal to 10 clients. In most cases, they are the worst. For this reason, we decide to skip doing the same experiments for other systems using different models and datasets. To have a started point, only one client was trained to obtain an initial result and compare it with the final model of the FL systems. The performance of this first model only depends on the number of clients in the system, during the experiments they are gonna be 5 and 20, both running on 10 epochs. The results from CNN and Transformers are in Table 5.7.

	CNN	Transformers
5 clients	19.56	23.15
20 clients	20.14	20.11

Table 5.8: One model trained for CNN and Transformer.

The models' hyperparameters are the default values that FAIRSEQ provides in its command lines. The values that we considered remarkable to mention are in Table 5.9:

Hyper-parameter	Tried values	Best value
Learning rate	0, 0.15, 0.25, 0.50	0.25
clip threshold of gradients	0.0, 0.10, 0.25, 0.50	0.10
Optimizer	adam, adamax, nag, sgd	nag
Dropout rate	0, 0.15, 0.20, 0.25, 0.50	0.20

Table 5.9: Tuning hyper-parameters of the model

The values of the hyperparameters were selected after the tests of the first experiments (from Tables 5.2 to 5.7). As we can see, the experiments were focused on CNN. However, we keep the hyperparameters' values for the Transformer due to the performance is acceptable.

The next results are focused on systems with 5 or 20 clients trained with 1 or 10 epochs using CNN (Tables 5.10 and 5.12) and Transformers (Tables 5.11 and 5.13) models. The systems described in the subsection FL systems are the cases described in the columns of the tables.

Clients	Epoch	*Case*1	*Case*2	*Case*3	*Case*4	*Case*5	*Case*6
5	1	21.17	21.17	19.92	20.04	20.04	19.92
5	10	23.99	25.28	23.32	23.97	24.60	23.32
20	1	19.39	19.39	19.53	20.55	20.55	17.11
20	10	21.11	23.50	22.39	22.78	23.79	20.57

Table 5.10: CNN trained with TED ES-EN dataset.

Clients	Epoch	*Case*1	*Case*2	*Case*3	*Case*4	*Case*5	*Case*6
5	1	22.29	22.29	21.53	21.45	21.45	21.53
5	10	23.84	25.04	23.17	24.99	26.09	23.17
20	1	21.38	21.38	21.54	22.43	22.43	18.68
20	10	22.58	22.72	21.89	23.85	24.92	20.37

Table 5.11: Trasnformer trained with TED ES-EN dataset.

Clients	Epoch	*Case*1	*Case*2	*Case*3	*Case*4	*Case*5	*Case*6
5	1	18.49	18.49	17.58	18.12	18.12	17.58
5	10	21.50	21.38	19.76	19.89	20.44	19.76
20	1	18.36	18.36	17.30	18.31	18.31	15.38
20	10	19.93	21.20	18.55	19.19	19.75	16.83

Table 5.12: CNN trained with TED NL-EN dataset.

Clients	Epoch	*Case*1	*Case*2	*Case*3	*Case*4	*Case*5	*Case*6
5	1	19.04	19.04	19.13	19.25	19.25	19.13
5	10	19.79	22.65	22.74	23.11	24.01	22.74
20	1	19.09	19.09	18.92	19.56	19.56	16.59
20	10	19.17	22.14	19.11	20.87	23.79	20.57

Table 5.13: Transformer trained with TED NL-EN dataset.

The final experiments are a sum-up of this work. The next table shows the best FL strategy for some clients in the system. In this case, the assumption is that each system has 5, 20, and 70 devices in the system training in 10 epochs. The model used is CNN trained with the TED ES-EN dataset.

Number of clients	Best strategy	Result
5	One epoch at a time	25.28
20	One epoch at a time with two clients	23.79
70	One epoch at a time	21.54

Table 5.14: Best strategy in a FL system

Chapter 6

Discussion

In this section, we discuss the results of the experiments explained before. We focus in to examine the results from systems based on CNN and Transformers models from Tables 5.10 to 5.13.

After running the experiments from Tables 5.2 and 5.7 we focus the experiments using one and ten epochs on systems with five and twenty clients and different strategies. As an expected outcome, every system with more than one epoch is more effective than the one trained with a single epoch, regardless of the model used in the experiment. Also, the results from a client trained only with its data (Table 5.7), which means not using FL, is not better than any FL system (Tables 5.10 to 5.13). The differences between the systems explained in Section 4 are described in the next lines. For an easier understanding of the strategies, they are going to be grouped into three.

The first one is oriented on sequential training: Sequential training (case 1) and one epoch at a time (case 2). The performance of both of them is medium compared with other cases. It should be noted that in systems' performance is the best one when it works with five clients and a single epoch instead of twenty clients. In a real scenario, it is not possible to select the number of clients trained in the system, but for the sake of this study, we assume that the number of clients is five during the whole training time. The performance obtained in these two scenarios is because the data in each client is large enough to bring acceptable results in the first clients.

The second group is focused on parallel training: Five clients in parallel (case 3) and FedAVG (case 6). The result for systems with five clients working at the same time and one epoch is acceptable comparing the other results, sometimes is even better, as we can see in Table 5.11. The worst performance is FedAVG for 20 clients. This is an expected result due to each client trains with a reduced dataset, which means that the average of the weights does not improve the resulting model.

Lastly, the third one is a combination of both of them. Sequential training with two clients in parallel. In most systems, these combinations are the ones with the best results. As we can see in Table 5.11, case 5 working with 5 clients

and 10 epochs is the finest result of this study. Finally, the baseline is better with Spanish than Dutch for both models. The reason is that there is more data in the Europarl dataset for Dutch than Spanish for training the model. As a result, the model performance is lower in Dutch due to it learns to translate from a formal language, but, conversely, it is tested with a more informal dataset (TED). As a result, the performance of the models is better when they have trained again with the TED Spanish dataset than Dutch because TED Spanish is larger than the Dutch one, as we can see in Table 4.1.

Chapter 7

Conclusions

Overall, the six FL systems using NMT scores had different results. Each of them is focused on parallelization, or sequential training. The Neural Network models used in this study are CNN and Transformer trained with the Europarl and TED dataset for Dutch and Spanish.

The main limitation of the FL systems is that models perform poorly on rare words due to the vocabulary used for the NMT models. The reason is that the Europearl dictionary is adopted to preprocess the TED dataset. This step is critical to avoid size errors during the model training. Moreover, the baseline is better with Spanish than Dutch for both models because it has more data in the Dutch language. Even when we train the baseline with the TED dataset, the influence of a re-purposed task is appreciated. Therefore, the performance with the Spanish data is better than the Dutch.

It is important to mention that we assumed that every client has the same amount of data. In a real-life environment, it is unlikely to happen. One of the future works for this research is to run the experiments from section 5 with clients that have an unequal amount of data. Also, it would be interesting to test clients with no data, how good are the translations of a model that has not been trained with specific user data.

The datasets used in this study are romance and german language. Models' performance is influenced by the amount of dataset and the times that the system averaged weights. However, it would be interesting to compare the results using a different language, like Scandinavian.

Another point to remark is that due to the limitations of the machine where we run the experiments, it was not possible to train parallel models using FAIRSEQ. For this reason, we simulate the parallel transition in the code. It would be important to study how fast is parallel training among a sequential one. We estimate that it would be much faster due to each model would run on its device, using a GPU only for that task. Also, it is possible to apply different amounts of data from clients' datasets, as we mention in previous paragraphs, with parallel training.

The last future work pruposed in this study i to combine the suggestions

in the last paragraphs with a different architecture of CNN and Transformer. Adding more layers to models could result in different performances. The other model available in FAIRSEQ is the LSTM. The results should be no different than those obtained in this study.

We can conclude that the number of clients is relevant to the final performance. Depending on the system, as more clients, the more average weights the system has to calculate, which means the loss is higher. However, there are other parameters to take into account.

It is significant to mention the models' performance after being trained with another dataset, which is the intention of Transfer Learning. The fact that models were trained first with the largest dataset (Europarl) is crucial to achieving better results. It means that the knowledge can be leveraged from existing models instead of starting from scratch each time. By using Transfer Learning we also remove the need to train with a large data for every client's model. Training only one time with Europarl data reduces the training time, even if it is not the goal of the thesis.

To continue with the structure of this document, we answer the research questions that we mention in the beginning.

1. How does the number of clients influence the performance of a Federated Learning?

   In this study, the number of clients is related to the size of the dataset. So, more clients mean a short dataset for each client. Depending on the system, the number of clients could be relevant to optimal results. For instance, if we have more clients in strategy 3, the performance does not vary significantly. Moreover, the result is barely better using a system based on Transformer models with 20 clients than CNN. On the other hand, the epoch number is more relevant than the number of clients in every system using CNN or Transformer. As we can see in the experiments, every Federated Learning system with ten epochs returns better results than systems with only one epoch. This behavior is expected and not surprising because as more data, better chances to improve the models. Even if the performance is slightly better with a greater value than 10, we do not consider it into account because if the number of epochs is getting quite large, so is the risk of overfitting.

2. What is the impact on the performance of the size of the training set?

   To answer this question, we have to compare the two datasets used in this study. First, we have to take into account that the TED Spanish dataset is larger than the Dutch one. If we analyze the results from table 5.10 and 5.12, we can appreciate that every system of table 5.10 has better performance than 5.12, so we could confirm that the size of the dataset is relevant. However, it is not the only reason to be considered. As we described in the first research question, the number of clients is important in this study because it is related to the size of the dataset. Coming back to the importance of the size of the dataset, it depends on which of them

(Europarl and TED). In this case, the size of the Europarl Dutch dataset is a little bit bigger than the Spanish one, which is proportional to the baseline's performance. The bigger the training Europarl dataset is, the lower the performance. The reason is that the model was trained on a specific task, but the test part is a different one. Also, the TED Dutch dataset is smaller than the Spanish one, as we can see in Table 4.1.

Finally, focusing on the FL strategies, the conclusion that we find in table 5.14 is that for a system with five clients, the best strategy is One epoch at a time. We assume that the result is influenced by the few clients in the system, making very costly and not efficent the average of the models' weights . In the second case, One epoch at a time with two clients is the one that suits a system with twenty clients because the they take advantage of the average weights. The amount of clients is around 20, with 10 or 50, the second strategy return finer performances as we can see in tables 5.3 and 5.6.The last case is the system with seventy clients, one epoch at a time is again the strategy with the best performance. At this point, every approach has similar results due to the dataset is tiny, getting hard the training state in the models, doing all the strategies' results alike.

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[2] Kyunghyun Cho et al. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259* (2014).

[3] Jonas Gehring et al. "Convolutional sequence to sequence learning". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1243–1252.

[4] Andrew Hard et al. "Federated learning for mobile keyboard prediction". In: *arXiv preprint arXiv:1811.03604* (2018).

[5] W John Hutchins. "Machine translation: A brief history". In: *Concise history of the language sciences*. Elsevier, 1995, pp. 431–445.

[6] Jim Isaak and Mina J Hanna. "User data privacy: Facebook, Cambridge Analytica, and privacy protection". In: *Computer* 51.8 (2018), pp. 56–59.

[7] Salman Khan et al. "Transformers in vision: A survey". In: *arXiv preprint arXiv:2101.01169* (2021).

[8] Philipp Koehn et al. "Europarl: A parallel corpus for statistical machine translation". In: *MT summit*. Vol. 5. Citeseer. 2005, pp. 79–86.

[9] Philipp Koehn. "Neural machine translation". In: *arXiv preprint arXiv:1709.07809* (2017).

[10] Adam Lopez. "Statistical machine translation". In: *ACM Computing Surveys (CSUR)* 40.3 (2008), pp. 1–49.

[11] Shengxuan Luo, Huaiyuan Ying, and Sheng Yu. "Sentence alignment with parallel documents helps biomedical machine translation". In: *arXiv preprint arXiv:2104.08588* (2021).

[12] Brendan McMahan and Daniel Ramage. *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`. [Online; accessed 06-April-2017]. 2017.

[13] Brendan McMahan et al. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[14] Larry R Medsker and LC Jain. “Recurrent neural networks”. In: *Design and Applications* 5 (2001), pp. 64–67.

[15] Emilio Soria Olivas et al. *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques: Algorithms, methods, and techniques*. IGI Global, 2009.

[16] Myle Ott et al. “fairseq: A fast, extensible toolkit for sequence modeling”. In: *arXiv preprint arXiv:1904.01038* (2019).

[17] Emilio Parisotto et al. “Stabilizing transformers for reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7487–7498.

[18] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural machine translation of rare words with subword units”. In: *arXiv preprint arXiv:1508.07909* (2015).

[19] Yu Shiwen and Bai Xiaojing. “Rule-based machine translation”. In: *Routledge Encyclopedia of Translation Technology*. Routledge, 2014, pp. 224–238.

[20] Tao Sun, Dongsheng Li, and Bao Wang. “Decentralized Federated Averaging”. In: *arXiv preprint arXiv:2104.11375* (2021).

[21] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. “Pixel recurrent neural networks”. In: *International conference on machine learning*. PMLR. 2016, pp. 1747–1756.

[22] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[23] Barret Zoph et al. “Transfer learning for low-resource neural machine translation”. In: *arXiv preprint arXiv:1604.02201* (2016).