UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

FEDERATED DEEP LEARNING FOR SENTIMENT ANALYSIS

Diploma Thesis

Nikolaos Karageorgos

Supervisor: Dimitrios Katsaros

Volos 2022

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

FEDERATED DEEP LEARNING FOR SENTIMENT ANALYSIS

Diploma Thesis

Nikolaos Karageorgos

Supervisor: Dimitrios Katsaros

Volos 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΟΜΟΣΠΟΝΔΗ ΒΑΘΙΑ ΜΑΘΗΣΗ ΓΙΑ ΑΝΑΛΥΣΗ ΣΥΝΑΙΣΘΗΜΑΤΩΝ

Διπλωματική Εργασία

Νικόλαος Καραγεώργος

Επιβλέπων: Δημήτριος Κατσαρός

Βόλος 2022

Approved by the Examination Committee:

Supervisor **Dimitrios Katsaros**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Dimitrios Rafailidis**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Georgios Thanos**
Laboratory Teaching Staff, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

I would like to thank my supervisor, Associate Professor Dimitrios Katsaros, for his constant guidance and valuable advice throughout this Thesis. I would also like to thank my friends and family for their support throughout my studies.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Nikolaos Karageorgos

Diploma Thesis

FEDERATED DEEP LEARNING FOR SENTIMENT ANALYSIS

Nikolaos Karageorgos

Abstract

Nowadays social networks are on the rise, so a big volume of data is produced. The sentiment analysis of this data has become a powerful means of learning about users' opinions and has a wide range of applications. However, in order to build an efficient model for that purpose, a large amount of data is needed and corporations are not always willing to share their data due to user privacy issues. Federated learning resolves this problem by enabling multiple client devices to collaboratively train a machine learning model while keeping their data private locally. In this Thesis, we simulate a decentralized environment where textual reviews of 3 different sources (IMDB, Amazon, Yelp) are stored in different nodes and collaboratively learn a sentiment classification model. For this purpose, we utilize the Federated Averaging algorithm which is a client-server federated learning approach where a central server aggregates the client model updates in order to construct the global model of the system. Three different deep learning models were utilized to test our approach: a Multilayer Perceptron with a global average pooling layer, a Bidirectional LSTM and a pretrained word emdedding model. The last model outperformed the two other ones in most cases but not with significant differences. The results of our experiments were quite satisfactory despite the data heterogeneity and convincing that such approaches could work in real world scenarios especially when there are enough data to reinforce the efficiency of the model so that it can overcome possible barriers and limitations.

Διπλωματική Εργασία

ΟΜΟΣΠΟΝΔΗ ΒΑΘΙΑ ΜΑΘΗΣΗ ΓΙΑ ΑΝΑΛΥΣΗ ΣΥΝΑΙΣΘΗΜΑΤΩΝ

Νικόλαος Καραγεώργος

Περίληψη

Στη σημερινή εποχή όπου τα δίκτυα κοινωνικής δικτύωσης βρίσκονται σε μεγάλη άνθηση, ένας τέραστιος όγκος δεδόμένων δημιουργείται. Η ανάλυση των συναισθημάτων αυτών των δεδομένων έχει γίνει ένα ισχυρό μέσο για την εξόρυξη πληροφορίων σε σχέση με την γνώμη των χρηστών και έχει αποκτήσει ένα ευρύ πλήθος εφαρμογών. Ωστόσο, για να δημιουργηθεί ένα αποτελεσματικό μόντελο για αυτό το σκοπό χρείαζεται ένα μεγάλο ποσοστό δεδομένων και οι εταιρείες δεν είναι πάντα διατεθειμένες να μοιραστούν τα δεδομένα τους για λόγους ιδιωτικότητας των χρηστών. Η Ομόσπονδη Μάθηση (Federated Learning) λύνει αυτό το πρόβλημα καθώς επιτρέπει σε πολλαπλούς πελάτες να εκπαιδεύσουν συνεργατικά ενα μοντέλο μηχανικής μάθησης κρατώντας τα δεδομένα τους ιδιωτικά τοπικά. Σε αυτή την εργασία προσομοιώνουμε ένα απογκεντρωμένο περιβάλλον όπου κριτικές από 3 διαφορετικές πηγές (IMDB, Amazon, Yelp) αποθηκεύονται σε διαφορετικούς κόμβους και συνεργατικά δημιουργούν ένα μοντέλο για την κατηγοριοποίηση των συναισθημάτων αυτών των κριτικών. Για αυτό το σκοπό, χρησιμοποιείται ο αλγόριθμος Federated Averaging ο οποίος είναι μια client-server προσέγγιση ομόσπονδης μάθησης όπου ένας κεντρικός server συλλέγει τις ενημερώσεις των τοπικών μοντέλων απο τους πέλατες (clients) για να δημιουργήσει το ενιαίο μοντέλο του συστήματος. Τρία διαφορετικά μοντέλα βαθιάς μάθησης χρησιμοποιούνται για ελέγξουμε την προσέγγιση μας: ένα Multilayer Perceptron με ένα global average pooling layer, ένα bidirectional LSTM και ένα μοντέλο με προεκπαιδευμένο embedding layer. Το τελευταίο μοντέλο ξεπέρνα σε επίδοση τα υπολοιπά δύο αλλά όχι με μεγάλη διαφορά. Τα αποτελέσματα των δοκιμών μας ήταν αρκετά ικανοποιητικά παρά την ετερογένεια των δεδομένων και πειστικά πως τέτοιες προσεγγίσεις θα μπορούσαν να δουλέψουν σε περιπτώσεις πραγραματικού κόσμου, ειδικά όταν υπάρχουν πολλά δεδομένα διαθέσιμα για ενισχύσουν την αποτελεσματικότητα του μοντέλου ώστε να μπορέσει να ξεπεράσει πιθανά εμπόδια και περιορισμούς.

Table of contents

List of figures

List of tables

Abbreviations

e.g.	exempli gratia
etc.	et cetera
NLP	Natural Language Processing
AI	Artificial Intelligence
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
ML	Machine Learning
DL	Deep Learning
FL	Federated Learning
TFF	TensorFlow Federated

Chapter 1

Introduction

1.1 Thesis Subject

Sentiment analysis is a process for determining a person's opinion, attitude, or feeling about a given issue. Text analytics and natural language processing are used in sentiment analysis to detect and extract subjective information from sources. Interest in sentiment analysis has grown as a result of the emergence of social networks where users are allowed to exchange opinions and share their thoughts about any topic. They may, for example, voice their dissatisfaction about a product they acquired, discuss current events, or express their opinions about politics. Business people usually rely on ratings, reviews recommendations and other kinds of online opinion in order to spot potential opportunities and maintain their reputations [1]. Furthermore, governments value public opinion analysis because it explains human behavior and how people are influnced by others' views. In addition, the inference of user sentiment can be highly beneficial in the area of recommender systems to compensate for the lack of clear user feedback on a delivered service. All the above emphasize the importance of an efficient model for sentiment classification and explains why AI researchers have focused their interest on this domain. However, in order to build a robust machine learning model a big amount of data is required.

In the big data era, more and more attention is payed to user's privacy and data protection. Data security has become a priority for both enterprises and individuals. Furthermore, in recent years, data leaking has piqued the interest of governments and the public media. Internet companies must not tamper or divulge with the personal data they receive from users, and they must verify that both the Internet company and the third party comply with user data protec-

tion standards while performing data transactions with third parties [2].As data privacy laws in many countries become stricter, large-scale user sensitive information exchanges between different organizations will no longer be permitted in the future. On the one hand, the adoption of these laws and regulations safeguards users' privacy, while on the other, it prevents large data from being mined arbitrarily, limiting the development of artificial intelligence.

The above problem is addressed by federated learning which is a machine learning method in which the training data is kept private and never leaves the client device In contrast to standard Machine Learning (ML), where all data is shared centrally, FL uses only locally computed updates that are communicated to the server by each device. The server integrates these updates into a final global model. This method is praised for its ability to maintain privacy. When dealing with the real-world issues of heterogeneous data and devices, it relies on distributed machine learning principles but goes beyond them in terms of privacy and performance. There are a number of reasons why FL is gaining popularity among consumers and enterprises. One factor is that this technique has the potential to reduce data privacy concerns. The widespread availability of modern computing machines is a second cause (e.g. mobile phones, tablets). A third reason is the advancements in Deep Learning, which are now available to data-sensitive sectors thanks to FL [3].

In this Thesis, we store in different nodes of a decentralized environment textual reviews of 3 different sources (IMDB, Amazon, Yelp) which could simulate 3 enterprises who want to build collaboratively a sentiment classification model without exchanging their data.For this purpose, we utilize the Federated Averaging algorithm which is a client-server federated learning approach where a central server averages the client model updates in order to build the global model. Three different deep learning models were utilized to test our approach: a Multilayer Perceptron with a global average pooling layer, a Bidirectional LSTM and a Pretrained word emdedding model.

1.2 Thesis Structure

This Thesis is organized as follows. In Chapter 2 is provided an extended introduction to Deep Learning and Federated Learning. Chapter 3 discusses several related works, the dataset compilation and preprocessing. Moreover, in this chapter our proposed neural network models for sentiment analysis and our chosen federated learning protocol are introduced.

Chapter 4 presents our experiments and analyses their results. Finally, Chapter 5 summarizes our work and proposes some directions for future work.

Chapter 2

Background

2.1 Sentiment Analysis

Sentiment analysis, also called as opinion mining, is the systematic identification, extraction, quantification and study of emotional states and subjective information using text analysis, natural language processing (NLP), computational linguistics and biometrics. Sentiment analysis is frequently used in customer service, marketing and healthcare to examine the voice of the customer items such as survey replies, reviews, social media and healthcare resources [4].

A common task in sentiment analysis, which we will be examining in this thesis, is the process of categorizing a text's polarity at the document, phrase, or feature/aspect level - whether it expresses a favorable, unfavorable, or neutral view. Advanced "beyond polarity" sentiment classification examines emotional states such as satisfaction, rage, sadness, worry, disgust and surprise.

The popularity of social media platforms such as blogs and social networking sites has increased interest in sentiment analysis. Online opinion has become a kind of virtual currency for firms wanting to advertise their products, uncover new prospects, and manage their reputations as a result of the increase of recommendations, reviews, ratings and other forms of online materials. Many enterprises have began to utilize sentiment analysis to in order to automate the task of separating out noise, comprehending discussions, selecting relevant content, and acting on it [5].

Statistical methods, knowledge-based strategies, and hybrid approaches are the three primary categories of sentiment analysis approaches. Knowledge-based strategies use unam-

biguous affect words like happy, sad, fearful, and bored to classify text into affect categories. Some knowledge sources not only list apparent affect terms, but also assign random words to emotions based on their ”affinity”. Statistical methodologies involve machine learning techniques such as semantic space models or word embedding models, latent semantic analysis, support vector machines and deep learning methods which we use in this thesis.

2.2 Deep Learning

Deep Learning (DL) is a subtype of Machine Learning (ML) which is a subtype of Artificial Intelligence (AI) Figure 2.1. AI is a field of Computer Science which examines the ability of machines to learn and perform tasks by simulating cognitive skills that are associated with the human brain. In such manner, computers are able to do jobs that are normally in need of human intelligence [6] AI employs both Machine Learning and Deep Learning methods to accomplish its goals.

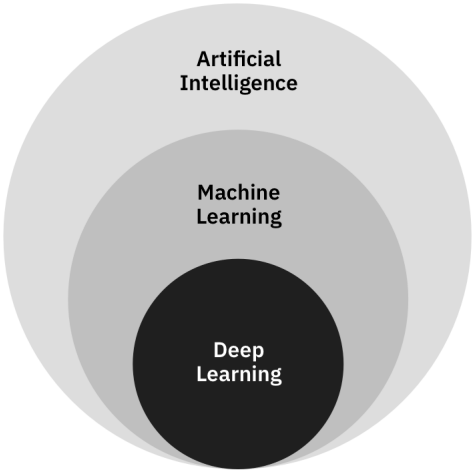


Figure 2.1: ML and DL as subsets of AI

Machine Learning is a field of study dedicated to understanding and developing ’learning’ processes which construct models utilizing training data in order to make predictions or conclusions without being expressly coded to do so. In some ways, the distinction between Deep Learning and Machine Learning is hazy. Deep Learning is a subset of a larger family of machine learning techniques based on artificial neural networks. These methods are

much more complex than normal ML approaches, as the term ”deep” refers to the depth of network’s layers, and often require massive amounts of data.

Deep learning utilizes representation learning which can be:

- **Supervised**

  Supervised learning utilizes matched inputs and desired outcomes (labeled data). For every input, the learning task is to deliver the expected output. The mean-squared error is a common cost aimed at lowering the average squared error between the inputs and the targets of the model. Suitable tasks for supervised learning include classification and regression. This is similar to learning with a ”teacher” in the manner of a function that gives feedback frequently about the performance of the solutions obtained so far. The problem we address in this thesis utilizes supervised learning as it is a classification task and the data is labelled; reviews of various contexts that have a positive or negative sentiment.

- **Unsupervised**

  The term ”unsupervised’ means acting without guidance or supervision and this refers to the lack of labeled data. In unsupervised learning untagged data get analyzed in order to extract patterns based on similarities or distinctions between them and in this ways they are clustered into groups. Input data is provided with a cost function, some function based on the data and the network’s outcomes. The cost function is determined by the problem (model domain) and any assumptions made in advance (properties, parameters and variables of the model). Clustering, filtering, compression and the estimate of statistical are all examples of unsupervised learning tasks.

- **Reinforcement**

  Reinforcement learning is interested in how intelligent beings should conduct in a given environment to optimize the concept of cumulative reward. Such algorithms are centred on encouraging desired moves and penalizing unwanted behaviors. A reinforcement learning agent acts and learns through experimentation (trial and error); e.g. an algorithm can taught by continually playing a game until it reaches the game’s highest marks. This type of learning is utilized in many areas such as information theory, game theory, multi-agent systems and simulation-based optimization

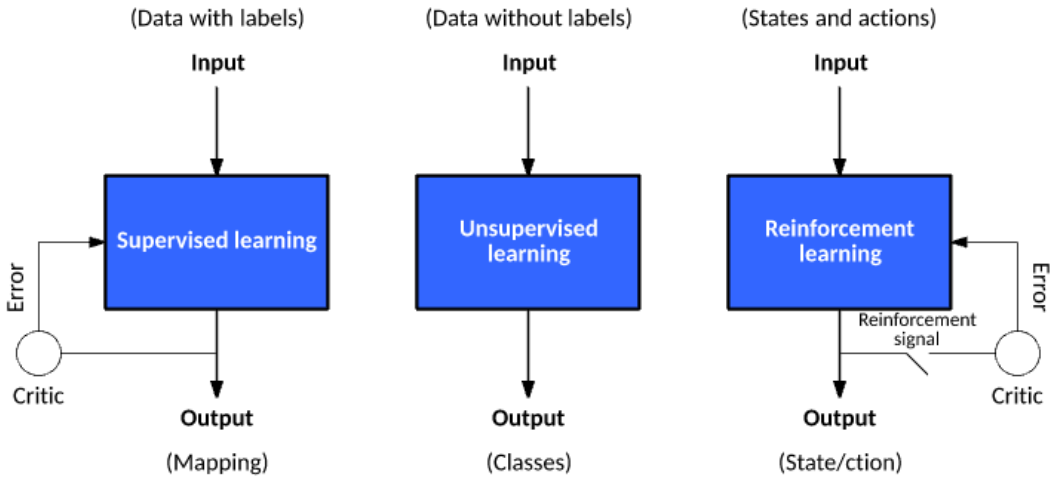


Figure 2.2: Representation learning schemas

2.2.1 Neural Networks

The terminology Deep Learning is strongly connected to Artificial Neural Networks (ANNs). ANNs were firstly introduced in the 1940s by Walter Pitts and Warren McCulloch who suggested computing models that simulated biological neural networks [7].The initial goal of the neural network technique was to develop a computing system that could address problems in the same way as a human mind could. However, as time went on, researchers began to focus on employing neural networks to fit certain tasks, deviating from the purely biological perspective.

Neural networks have since been capable of helping people in real-life scenarios with complex tasks. They can be trained and learn nonlinear and complex relationships between inputs and outputs, draw inferences and generalizations, discover hidden links, predictions and patterns, model data of high volatility and variances to forecast unusual events.As a follow-up, they have been used to serve a variety of tasks e.g.computer vision, natural language processing, speech recognition, social network analysis and medical diagnosis [8]. But how exactly do ANNs work?

ANNs along with their architecture, as mentioned above, are derived from the human brain and they simulate how actual human neurons work and interact. Their basic structure consists of an input layer, one or more hidden layers in between and an output layer as depicted in Figure 2.2. In case there are more than one hidden layers in the ANN, it is characterized as a Deep Neural Network [9]. These layers consist of artificial neurons and try to capture correlations and patterns between the data they have to examine. These neurons, also

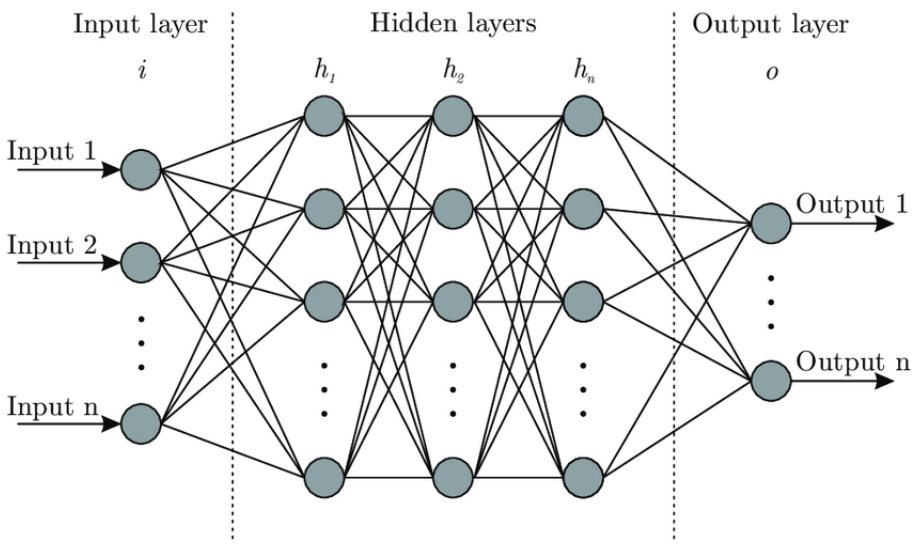


Figure 2.3: Architecture of ANNs

called as nodes, are the network's central processing units. Through the input layer, which connects to the hidden layers, the data is fed into the neural network. An artificial neuron is a small unit that is linked to the others and has a weight and a threshold associated with it. Input signals are transmitted to the hidden layers and are multiplied by the weights of the links they pass through. The hidden layers perform some transformations and adjustments using the activation functions and in such manner they decide how much a signal should continue propagating into the network so as to impact the final outcome [10] .

The connections between nodes are weights which can take negative or positive values. This represents how important effect has the input of the previous node on the outcome of the following node. Updating the weights of a model is the principal technique for the learning process of a neural network. Features with weights which have values close to zero seem to have less influence on the prediction task in comparison to features with weights that have higher values. A neural network is taught when provided with feedback about whether it made the right adjustments or not. The training can be terminated when predefined criteria have been met after enough of these modifications. [11].

Activation functions

Activation functions, also called as Transfer functions, are an essential component of neural network's structure. They define the neuron's output. The output of the activation function is called forward propagation. Activation functions perform arithmetic operations in

order to decide whether a neuron's output is essential to the network and needs to be activated. The primary purpose is to turn the cumulative weighted input of the neuron into an output value that may be forwarded to the next hidden layer or output of the model. The activation function employed in the output layer determines the type of predictions the network can make.

Activation functions can be divined into to two main categories which are Linear Activation Function and Non Linear Activation functions.

- **Linear Activation Function**

  The linear activation function (or else identity function) is the function in which the activation is proportionate to the input (weighted sum of the nodes). It makes no adjustments to the input and just gives the value it was given. It has a standard form with the following formula: $f(x) = ax + b$ (shown schematically in Figure 2.4). In case this activation function is used at all nodes, it behaves like linear regression [12].

  As a form of the linear activation function is also classified the binary step function which is given by the formula:

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{2.1}$$

- **Non-Linear Activation Functions**

  The most commonly utilized activation functions are non-linear functions.The primary element that make them efficient is that they are differentiable and can address the Vanishing Gradient problem. They make it simple for a neural network model to adjust to diverse types of data and distinguish between distinct outputs. Some primary non-linear activation functions are presented below:

  - **Sigmoid**:

  The Sigmoid Function curve is shaped like the letter S as it is derived from its name. They are widely used for networks where probabilities need to be predicted as outputs. They are suitable for this task as they return values in the range 0 to 1 which is the same range of the values that probabilities take [13]. The sigmoid function is defined by the formula:

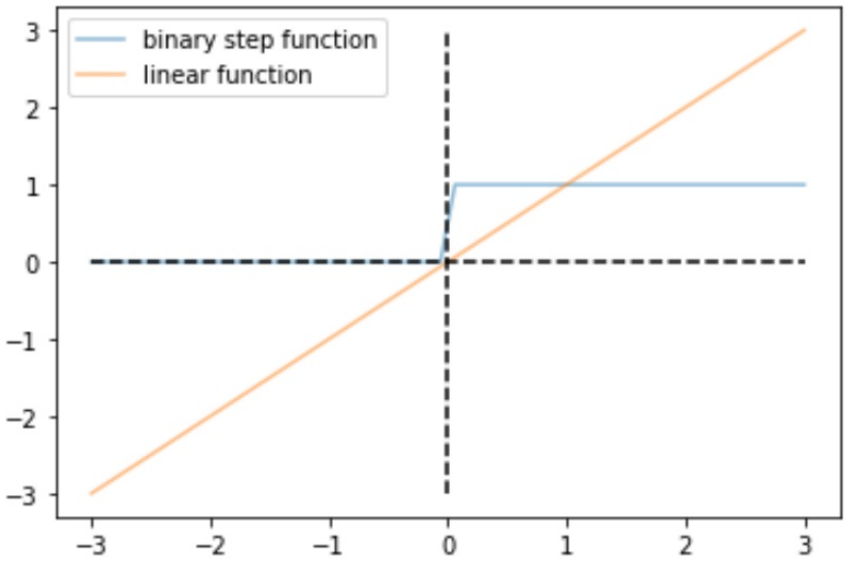


Figure 2.4: Linear activation functions

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

It's a differentiable function with a smooth gradient from which it is observed that only the gradient values in the range -3 to 3 are meaningful, and the graph becomes significantly flatter in other regions. This signifies that the function will have very weak gradients for values less than -3 or more than 3. The network has been taught well when the gradient value approximates zero.

- Tanh:

The Tanh function, also called as hyperbolic tangent, is quite similar to the sigmoid activation function, and also has the S-shape with a -1 to 1 output range difference.The output is closer to 1 as greater is the positive input value; the closer the output to -1.0 as smaller is the negative input value. It is defined for all real input data and has a positive derivative at each point. The tanh function is defined by the formula:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.3}$$

- ReLU:

ReLU (Rectified Linear Unit) differs mainly from other activation functions because it does not activates all of the neurons simultaneously. This means that the neurons will only be muted if the outcome of the linear function is less than 0. It converges considerably faster than the sigmoid and tanh functions because just a little number of neurons stay active. Therefore,ReLU is the most extensively utilized activation function, especially for hidden layers [14]. Leaky ReLU and Parameterised ReLU are versions of the ReLU function that address specific problems that the original function cannot handle efficiently. The tanh function is defined by the formula:

$$f(x) = x^+ = \max(0, x) \tag{2.4}$$

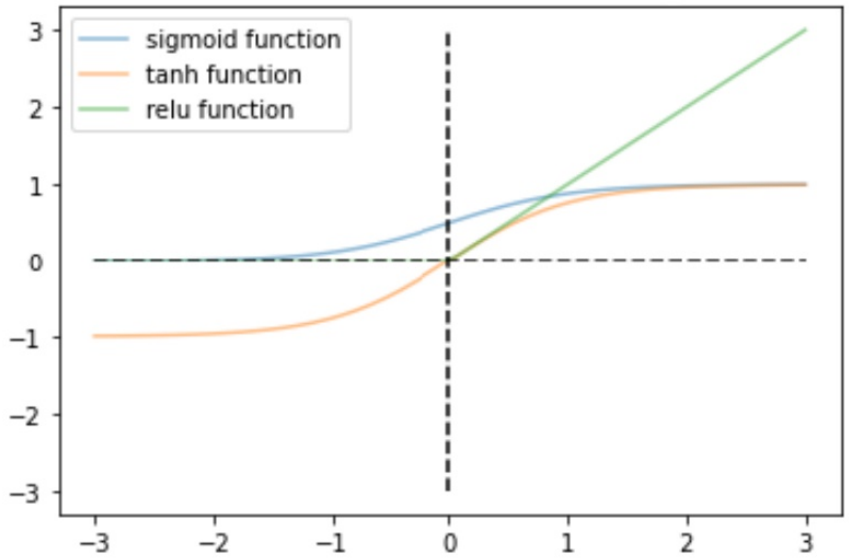


Figure 2.5: Non-Linear activation functions

Loss functions

Neural networks utilize loss functions in their process of learning. By using them they can determine how accurately a certain algorithm fits the data. If predictions differ significantly from actual findings, a large value is returned by the loss function. Some optimization function helps loss function to learn to decrease prediction error over training time. [15]. The loss function is used to compute the gradients that are utilized to adjust the parameters of the

neural network. It's how the Neural Network gets trained, and it's how we determine how well or inadequately the model is doing.

In terms of the types of problems we encounter in the actual world, loss functions can be divided into two categories: classification and regression. Our objective in classification challenges is to forecast the respective probability of all classes included in the problem. Common functions for classification tasks are Cross-Entropy based Loss Functions. As far as regression is concerned, on the contrary, the goal is to predict the continuous value for a given set of independent features using a learning method. For regression problems the most common utilized functions are the Mean Absolute Error (MAE) and the Mean Squared Error (MSE) [16].

Optimizers

The concept of loss indicates how well a model performs, as it was mentioned above. Optimization is the process of minimizing the loss so that the model works better. Optimizers are techniques for adjusting the parameters of a neural network in order to reduce losses. By reducing the loss function, optimizers are used to address optimization problems.

There are plenty different types of optimizers but some of the most primary ones that were overviewed in this project are:

- **Gradient Descent:** The most common yet extensively used optimization approach is gradient descent. It's often utilized in methods for classification and linear regression. Backpropagation process in neural network uses also the gradient descent algorithm.

  Gradient descent is a first-order optimization method which relies on the loss function's first order derivative. It determines in which direction the weights should be altered in order for the function to reach a minimum. The loss is passed from one layer to the next through backpropagation and the parameters of the models are updated when the gradients of the entire data are computed [17]. This means that if the data is massive this process can last for a long amount of time.

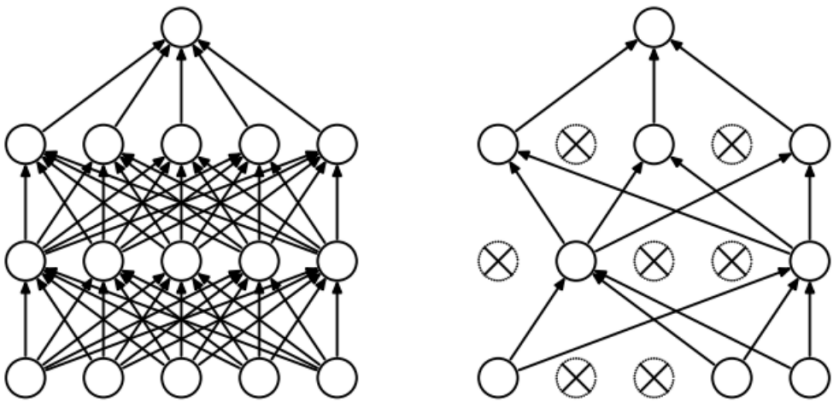
- **Stohastic Gradient Descent:** It is a variation of Gradient descent. It attempts to adjust the parameters of the network more often. This adjustments are made when each training sample's loss has been calculated. In such manner, this optimization method reaches convergence faster [18].

- **Adam**: This optimizer keeps track of each network parameter's learning rate and adjusts it independently throughout training. It adapts the learning rate for each weight depending on estimated first and second gradients [19].

- **Adagrad**: This optimizer adjusts the learning rate to the parameters, making smaller updates for frequently appearing features and larger updates for rarely occuring features [20]. This makes Adagrad a suitable optimizer for dealing with sparse data.

Dropout

During the training of a randomly selected group, dropout describes the act of ignoring neurons. With "disregarding," we mean that a neural network is forced to develop more resilient properties that can be applied to several random subsets of other neurons. During a forward or backward pass, these units are not taken into account [21]. It's a technique for reducing overfitting. A model suffers from overfitting when a model works admirably on the training data, but not so well on the evaluation data. Because large networks take time to build, it's difficult to avoid overfitting by combining predictions from multiple neural networks at test time. This is a problem that Dropout is grappling with. This approach has been identified to improve neural net performance in a variety of applications. In our models in this work we employ dropout layers to avoid overfitting.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.6: Neural networks before and after applying dropout

2.2.2 Recurrent Neural Networks

Recurrent neural networks (RNNs), due to their internal memory, are the state-of-the-art method for time-series or sequential data. RNNs get their name from the fact that they apply the same function for every element of a sequence, with the output being reliant on previous calculations [22]. They contain loops that allow data to persist. The inputs are all interconnected together as it shown in Figure 2.6.

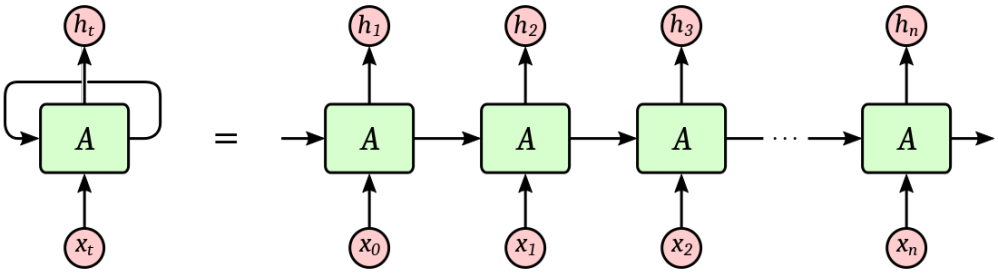


Figure 2.7: Unrolled RNN.

During training, however, typical recurrent neural networks encounter the problem of vanishing gradients. Values in a very lengthy sequence may have long-term dependence. Because of the nature of the training process, it's probable that a conventional recurrent neural network won't be able to learn these dependencies when memorizing these sequences. Creating versions of regular recurrent neural networks that change how these gradients are calculated is one approach to this challenge.

Long Short-Term Memory (LSTM)

The solution to the above problem of vanishing gradients is given by a improved variation of RNNs which are called Long Short-Term Memory (LSTM) networks. These networks make it easier to recollect knowledge from the past memory as the gradients are controlled by several gates in the long short term memory cell. These gates are the Input gate, the Output gate and the Forget gate (Figure 2.7).

The Input gate determines how the memory will be altered. The Sigmoid function will determine which data will be sent on to the next layers, while the Tanh function will provide a weight to the input based on its importance to the network. The Forget gate determines which data should be dismissed. The Sigmoid function is in charge of this procedure; it returns a number in the range [0, 1] by looking at both the previous state and the input. If the number

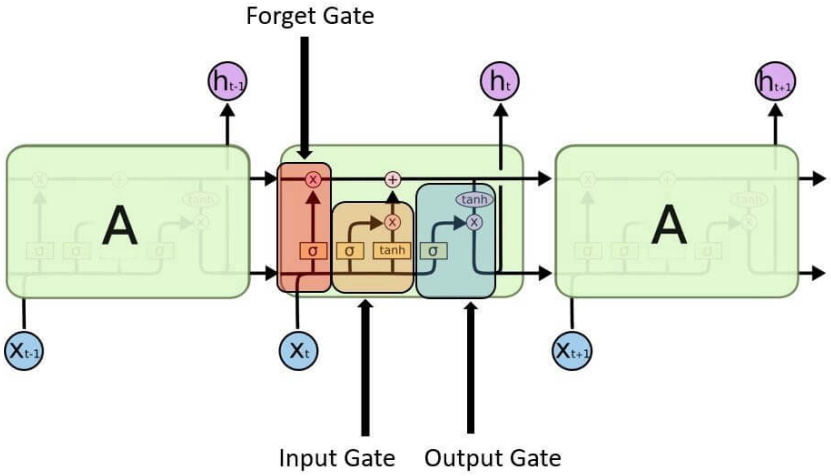


Figure 2.8: LSTM gates architecture.

is 0, it is omitted; however, if the number is 1, it is kept. Finally, the memory and the block's input will define the output gate. The Sigmoid function decides which values are allowed to pass, whereas the Tanh function gives weight to those that do [23].

In our project we utilize a Bidirectional LSTM. It differs from a traditional LSTM because it processes inputs in two directions, one from the past to the future and the other from the future to the past, allowing it to maintain information from both the future and the past..

2.2.3 Transfer Learning

Transfer learning is a ML research subject that entails storing knowledge obtained from one problem and using it in a another different but related task [24]. This knowledge can be either weights or embeddings.

The notion is known as a pre-trained model in the case of weights. Simply explained, a pre-trained model is one that has already been trained to handle a similar problem by someone else. Rather than starting from scratch to tackle a comparable problem, the model trained on the previous problem is used as a starting point. For instance, in the case of developing a self-learning automobile, many years could be spent in order to develop an efficient image recognition model from the beginning but it would be easier to utilize Google's pre-trained inception model which was trained on ImageNet dataset to detect images in those photographs. Although a pre-trained model might not be extremely efficient in a specific application, it

saves a big amount of effort and time by not having to address the problem from scratch. A way to fine-tune a pre-trained model is feature extraction. This method utilizes as a feature extraction mechanism a pre-trained model. For the new dataset it uses as a fixed feature extractor the complete network.It achieves that by removing the output layer. Another method is to use the entire architecture of the pre-trained model by initializing the weights randomly and then retrain the model using our dataset [25].

In the case of the embeddings, the notion is known as pre-trained word embeddings which we utilize in this project. As they are trained on massive datasets, pre-trained word embeddings capture the syntactic and semantic meaning of a word. They are able to improve a NLP model's results. These word embeddings are efficiently useful because they are difficult to learn from beginning for two primary reasons. The first one is the sparseness of training data as the majority of real-world problems include a dataset with a substantial number of infrequent words. These datasets' embeddings are not capable of producing the accurate representation of the word. To accomplish this, the dataset needs have a big vocabulary. Frequently recurring words help to create a diverse vocabulary. Second, the number of Trainable Parameters increases while learning word embeddings from the beginning. As a consequence, the training procedure takes longer. Learning embeddings from the ground up may leave uncertainty of how the words should be represented [26].

2.3 Federated Learning

Federated learning , often referred to as collaborative learning, is a machine learning method that entails training an algorithm across multiple decentralized servers or edge devices that keep local data private without exchanging them. This method is different from standard centralized machine learning methods, which all need local data samples to be compiled to a single central server, and it also differs from more traditional decentralized variations, which often presume that local datasets are identically distributed [27].

This technique allows numerous clients devices to develop a common, strong machine learning model without communicating their data, allowing crucial issues like data access rights, security, access to heterogeneous data and data pricacy to be addressed. Defense, IoT, telecommunications, and healthcare are just a several of the industries where it's applied.

Federated learning is utilizing numerous local data samples stored in local nodes to train

a machine learning algorithm, such as deep neural networks which we utilize to train in this project, without explicitly sharing data samples. The general idea is developing local models on local private datasets and periodically trading parameters (e.g. the network's weights and biases) between local nodes to produce a global model maintained by all nodes.

The principal distinction between distributed and federated learning is the hypothesis made about the features of local data, as distributed learning was designed to parallelize computing capacity, whereas federated learning was composed to be used for training on heterogeneous data samples. While distributed learning also tries to train a single model over numerous servers, a common implicit hypothesis is that the local data are independent and identically distributed (i.i.d.) and of nearly similar size. None of these concepts are applicable for federated learning because data are often heterogeneous and can vary in size by multiple orders of magnitude.

A federated learning system might or might not include a central coordinator-server and based on this difference it can be divided into to two primary categories:

- **Centralized federated learning (client-server model)**

  The coordinator in this case is a central aggregation server which is responsible for the communication process of training between the local nodes. The data of the local data owners does not leave the local nodes under this architecture.

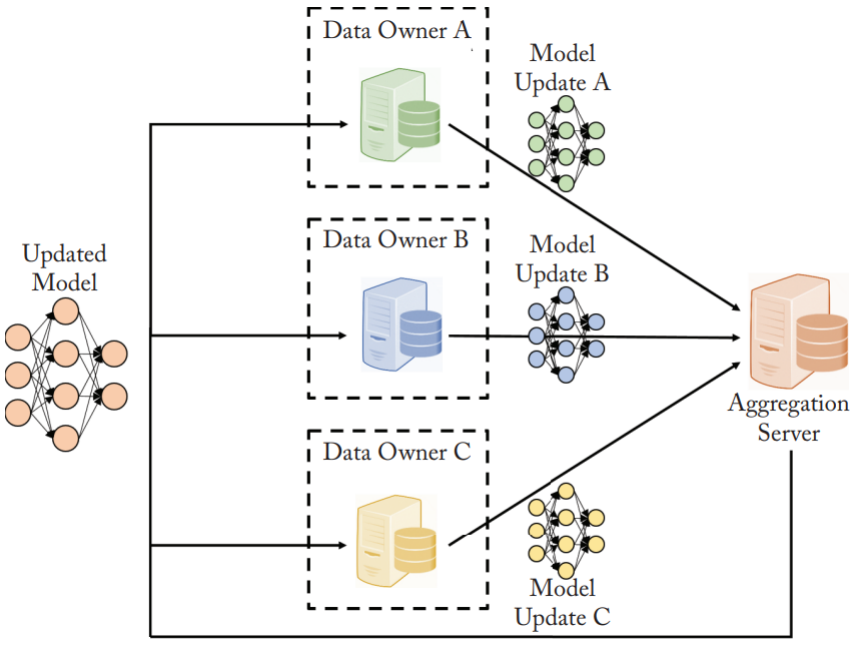


Figure 2.9: A client-server federated learning architecture example.

This method not only protects user privacy and data security, but it also reduces the amount of time it takes to transfer raw data. The communication between the central aggregating server and the local nodes can be encrypted. [28].

- **Decentralized federated learning (peer-to-peer model)**

  A peer-to-peer federated learning architecture can be created without the need for a coordinator. As shown in Figure 2.7, this provides an additional layer of security by allowing the parties to communicate directly without the help of a third party. Increased security is a benefit of this architecture, but the cost of encrypting and decrypting messages could be higher.

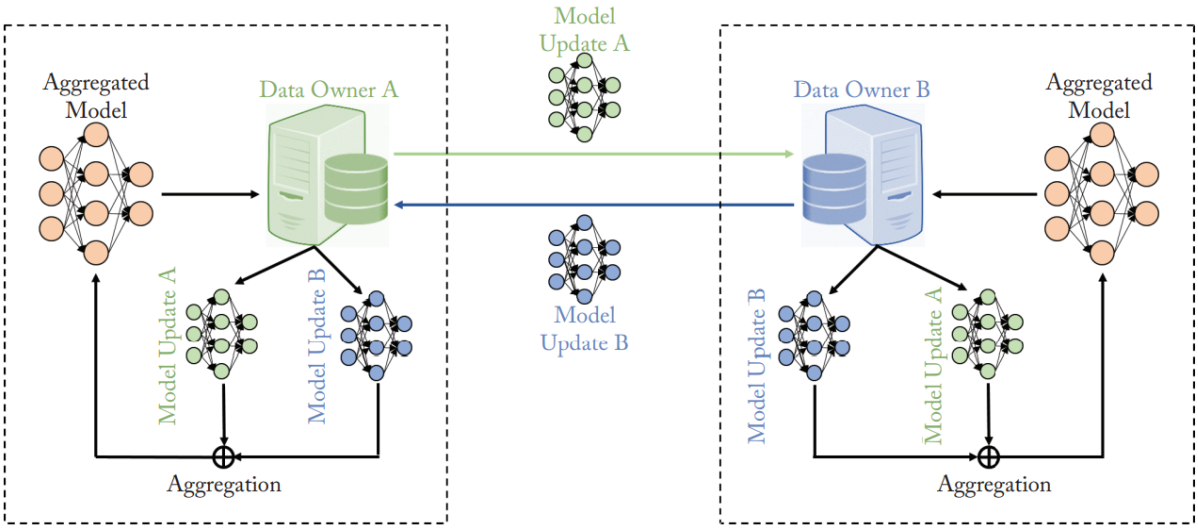


Figure 2.10: A peer-to-peer federated learning architecture example.

Iterative process

Federated learning is based on an iterative process, referred to as a federated learning round, which is divided up into an atomic collection of client-server interchanges to assure the efficiency of a final, global machine learning model. Every round of this process entails sending the current central model state to the nodes participating, training local models on these nodes to generate a set of possible model updates at every node, and finally combining these local updates into a single overall update and adapting it to the central model [29].

2.4 TensorFlow

TensorFlow is a free software platform for machine learning. It includes a flexible and large wide range of libraries, community resources and tools that allow developers to build fast and deploy ML model and academics to work on new advancements in ML [30]. TensorFlow offers a collection of workflows for creating and training applications in Python or JavaScript, and for deploying them in the browser, in the cloud, on-device or on-prem, no matter what language is used.

2.4.1 TensorFlow Federated

TensorFlow Federated (TFF) is an openly accessible framework for decentralized machine learning and other operations. TFF was designed to enable open study and experimentation with Federated Learning (FL), a machine learning method in which a shared global model is trained across several clients who maintain their training data privately. FL has been used to train forecasting models for mobile keyboards without having to submit sensitive typing data to servers, for instance [31].

TFF offers the possibility to developers to examine with new algorithms as well as simulate the integrated federated learning algorithms on their data and models. For various types of inquiry, researchers will find starting points and entire examples. Non-learning computations, such as federated analytics, can be implemented using TFF's building pieces.

In this project, we utilize the TFF framework to simulate our decentralized environment and implement the Federated Averaging algorithm.

Chapter 3

Methodology

3.1 Related Work

Theoretical work isn't the only thing that piques researchers' interest in federated learning. The development and deployment of federated learning algorithms and systems is likewise a burgeoning field of study.

Google firstly introduced federated learning (FL) in 2016, which has since been further improved by other studies [32]. The authors have already identified imbalanced and non-IID (independent and identically distributed) data, as well as a large number of participating devices with varied levels of trustworthiness and potentially significant transmission costs, as the field's defining problems. Over the last few years, the study field has been driven by questions and concerns about changing amounts of user data gathered from different distributions, differences in bandwidth and computational capability, as well as communication costs and privacy hazards [29].

Therefore, there are various federated learning open-source projects that are rapidly increasing and each one attempts to address a specific issue on federated learning. OpenMined/PySyft [33], TensorFlow-Encrypted [34] and Federated AI Technology Enabler (FATE) [35] focus on supporting methods of secure computation through homomorphic encryption. Other frameworks like Tensorflow-Federated [31], LEAF [36] and coMind [37], aimed towards capturing the realities, challenges, and complexities of practical federated learning environments, offer settings for simulating federated learning computations and aggregated analytics over decentralized data.

Many works have focused their interest on applying federating learning algorithms like

FedSGD [32] and FedAvg [38] on NLP problems such as language model training [39], mobile keyboard prediction [40], , spoken language understanding [41] and and news recommendation [42]. A relevant work to ours is that of [43] which uses a federated averaging variation to train LSTM models to perform sentiment analysis on tweets from a decentralized version of dataset Sentiment140 using LEAF. Another related study [44] proposes a client-server federated algorithm for sentiment classification of customers' reviews from different public benchmarks with the consideration of the huge embedding size properties of text vectors utilizing transfer learning. In this project, we are inspired by the paradigm from [45] to apply federated learning for sentiment analysis on three datasets with customers' reviews of different context using the Federated Averaging algorithm, where information, including model weights and losses, are exchanged between the server and clients.

3.2 Data Compilation

In terms of the reproducibility of outcomes, the datasets selection is critical. This motivated the selection of three of the most widely used datasets for NLP research. This choice of datasets defined the NLP task of our project which is binary sentiment classification. The datasets that were selected are:

- **IMDB**

  This dataset was obtained from Kaggle [46]. It contains 50,000 highly polarized movie reviews for binary sentiment classification. There is also supplementary unlabeled data available for use.

- **Amazon**

  This dataset was acquired from [47]. It is a massive dataset as it contains 34,686,770 Amazon reviews on 2,441,053 products from 6,643,669 users. It was compiled from he Stanford Network Analysis Project (SNAP). The subset that is labeled for binary sentiment classification includes 1,800,000 samples for training and 200,000 samples for testing in every sentiment category.

- **Yelp**

  This dataset was obtained from [48]. This is also a huge dataset as it includes about 7,000,000 online reviews of restaurants and hotels for sentiment analysis.

3.3 Data Preprocessing

Text preprocessing is typically an essential step for natural language processing (NLP) tasks. It converts text into a more suitable form so that ML or DL algorithms can perform better. We apply some basic preprocessing techniques for text reviews to our datasets following the paradigm from [49]. These techniques are:

- **Lowercase.** A common preprocessing method that converts all texts to lowercase. Because the identical words are fused, the problem's dimensionality is decreased.
- **Removal of numbers.** Numbers do not generally contain any sentiments, so they should be removed; nonetheless, this should be done after emotional icon substitution and wrongspelling correction because some of them contain numbers, such as 8-), :-3 etc.
- **Removal of punctuation.** Punctuation does not normally impact the sentiment, therefore it should be deleted to reduce noise. However, because some punctuation contains sentiment, it may reduce classification performance in those circumstances, such as :), ;), :D are positive icons that impact sentiment in reviews. This step will be used after the emoticon symbols have been replaced in our project.
- **Emoticons replacement.** Emotional icons have been commonly utilized in reviews to represent readers' feelings, so it is important to express the sentiment of them. The positive and negative icons are replaced with "pos" and "neg" lexicons in our scenario.
- **Removal of stop words.** Stop words are function words; they normally have little meaning and don't convey much emotion, but they appear frequently in texts. To reduce dimensionality and computational expense, as well as increase performance, they should be deleted. Depending on the application, the set of these words isn't completely predetermined. Stop words were determined in our project based on term frequencies and inverse document frequencies weights in the datasets obtained.

To apply these basic preprocessing techniques to our data we utilize a Text Vectorization layer. This layer after applying these standarizations is used to divine each input into substrings, rejoin substrings into tokens, index these tokens (assign a single int value with each token) and convert each sample utilizing this index into a vector of ints. With a method of

this layer we construct a vocabulary for the data that is being applied to. Depending on the setup parameters for this layer, this vocabulary can be limitless or capped; in case there are more single values in the input data samples than the maximum vocabulary size, the most frequently occuring phrases will be selected to generate the vocabulary. In our case we have set our vocabulary size to 10000 features. After aplying this Text Vectorization layer to the data, their representation can be read by an Embedding layer or Dense layer.

3.4 Neural network architectures for sentiment analysis

For our task which is binary sentiment classification of reviews we have chosen to train three different neural network models, a Multilayer Perceptron with a global average pooling as hidden layer, a Bidirectional LSTM and a Pre-trained word embedding model. These models will be trained on the data of each node where a different dataset is being stored, their parameters will be exchanged between the nodes for the federated learning process and their performance will be compared and discussed.

3.4.1 Multilayer Perceptron

Table 3.1: MLP architecture summary

Layer type	Output Shape	Parameters
Embedding	(None, None, 16)	160016
Dropout	(None, None, 16)	0
GlobalAveragePooling1D	(None, 16)	0
Dropout	(None, 16)	0
Dense	(None, 1)	17
Total trainable parameters: 160,033		

After the data is preprocessed as discussed in paragraph 3.3 is suitable to be fed into the neural network for training. In this neural network, to construct the classifier, the layers are placed sequentially in the following order:

1. An Embedding layer is used as the first layer. This layer looks up for each word-index

an embedding vector from the integer-encoded reviews. As the model trains, these vectors are created. A dimension is added to the output array by the vectors.

2. After that, a GlobalAveragePooling1D layer averaging across the sequence dimension gives a fixed-length output vector for every case. This enables the model to deal with variable-length input in the most straightforward manner feasible.

3. A fully-connected (Dense) layer which has 16 hidden units is used to transmit this fixed-length output vector.

4. The last layer has a single output node and is densely connected.

3.4.2 Bidirectional Long-Short-Term-Memory (LSTM)

Table 3.2: Bidirectional LSTM architecture summary

Layer type	Output Shape	Parameters
Embedding	(None, None, 64)	64000
Bidirectional LSTM	(None, 128)	66048
Dense	(None, 64)	8256
Dropout	(None, 64)	0
Dense	(None, 1)	65
Total trainable parameters: 138,369		

The data can be supplied into the neural network for training after it has been preprocessed as described in paragraph 3.3. This model is built in this order:

1. The first layer is an embedding layer. Each word is stored in an embedding layer by one vector. It converts sequences of word indices to vector sequences when invoked. These vectors can be learned. Words with similar meanings often have also similar vectors after training (on enough data).

   The analogous action of sending through a Dense layer a one-hot encoded vector is substantially more inefficient than this index-lookup.

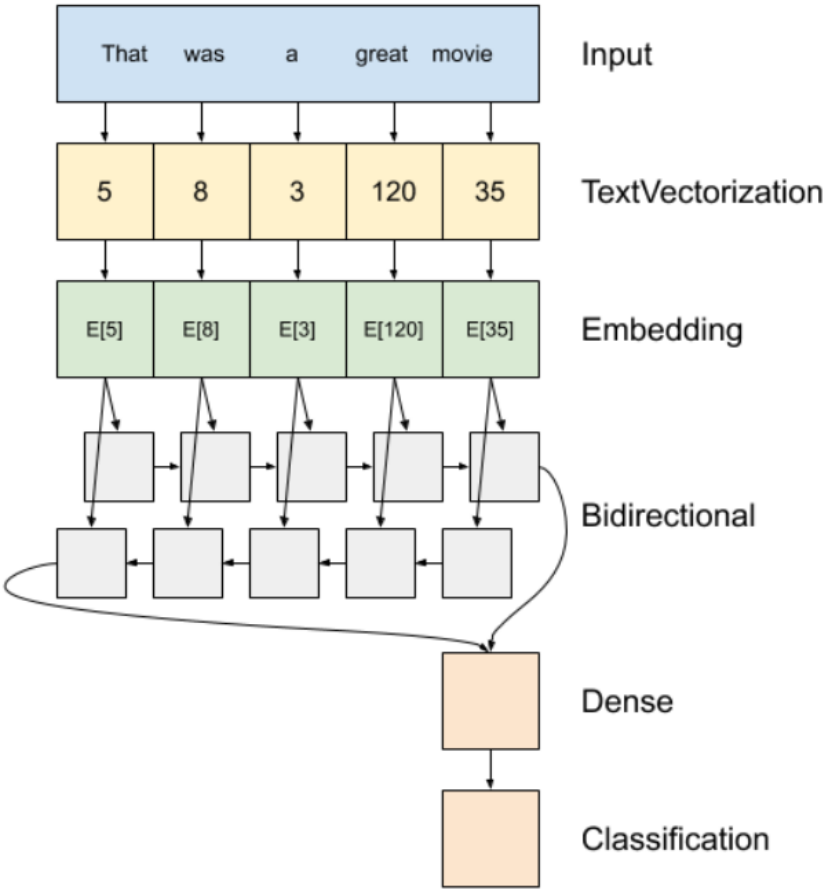


Figure 3.1: Bidirectional LSTM model diagram

2. By iterating through the elements, a recurrent neural network (RNN) handles sequence input. The outputs of one timestep are passed to the input of the following timestep by RNNs.

   The Bidirectional wrapper can also be utilized with an RNN layer. The input is propagated backwards and forward via the RNN layer, and then the output is concatenated.

   The fundamental advantage of a bidirectional RNN is that the signal does not have to be processed from the start of the input all the way through every timestep to impact the outcome. While the fundamental drawback of a bidirectional RNN is that it is difficult to stream predictions when new words are appended to the end.

3. The two Dense layers perform some final processing after the RNN has transformed the sequence to a single vector, then transform from this vector representation to a single logit as the classification output.

3.4.3 Pre-trained word embedding model

Table 3.3: Pre-trained word embedding model architecture summary

Layer type	Output Shape	Parameters
KerasLayer	(None, 50)	48190600
Dense	(None, 16)	816
Dropout	(None, 16)	0
Dense	(None, 1)	17
Total trainable parameters: 48,191,433		

The layers are stacked sequentially to build the classifier:

1. The first layer is a layer from TensorFlow Hub. This layer utilizes a pre-trained Saved Model to associate a sentence into its embedding vector. The pre-trained word embedding model that we use (google/nnlm-en-dim50/2) [50] divides the sentence into tokens, embeds each token separately, and then combines the embedding. The embedding_dimension is 50 for this model.

2. A fully-connected (Dense) layer that has 16 hidden units is used to route this fixed-length output vector.

3. With a single output node, the final layer is densely connected.

Using a pre-trained text embedding as the first layer has numerous advantages. We do not have to perform any text preprocessing as we did on the first two models (as discussed in paragraph 3.3) as we benefit from tranfser learning. Moreover, the embedding has a predefined size, so it is easier to process. On the other hand, one drawback of the pre-trained text embedding model is that we have a very large embedding dimension and many more trainable parameters as we have adopted a much larger vocabulary and this may improve our task but it can take much longer to train our model.

3.5 Federated Learning Process

The nodes are trained on their data and communicate their model parameter updates in every federated iterative learning round. In the methodology we have chosen to use, aggregation is handled by a central server, while local nodes undertake local training in response to the central server's commands.

The learning procedure can be summarized by the following five steps as follows, assuming a federated round constituted of one iteration of the learning process:

1. Initialization: One of the deep learning model we have introduced is chosen based on the server inputs and trained and initialized on local nodes. The nodes are then turned on and wait for the calculation assignments to be assigned by the central server.

2. Server-client model transfer: Local nodes are selected to begin training on local data and the current model is sent to these nodes.

3. Configuration: The central server instructs nodes to train the model on their local datasets according to a predetermined schedule.

4. Reporting: For aggregation, each node delivers its local model to the server. The model changes are sent back to the nodes by the central server, which aggregates the received models.

5. Termination: The central server combines the updates and constructs the final global model after a termination criterion is fulfilled (e.g., a maximum training rounds are reached or the accuracy surpasses a threshold).

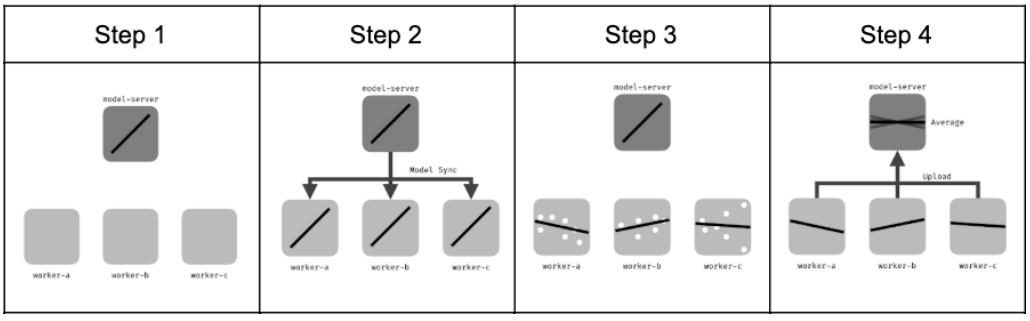


Figure 3.2: Federated learning general process in our setup

3.5.1 Federated Averaging algorithm

Particularly, we follow the above protocol utilizing the Federated Averaging (FedAvg) algorithm which was proposed by [38]. FedAvg is a variation of FedSGD that enables local nodes to execute multiple batch updates on local datasets and trade updated weights rather than gradients. The reasoning behind this generalization is that in FedSGD, averaging the gradients is strictly identical to averaging the weights themselves if all local nodes start from the same initialization. Furthermore, averaging tuned weights from the same starting does not always degrade the performance of the final averaged model.

Algorithm 1 The FEDERATED AVERAGING algorithm. The K clients are denoted by k; E is the number of local epochs, B is the local minibatch size and n is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

One important note about the Federated Averaging algorithm is that it has two optimizers: a client optimizer and a server optimizer. The client optimizer is only used on each client to perform local model updates. The server optimizer is utilized to adjust the global model with the averaged update.

Chapter 4

Experiments & Results

4.1 Experimental set-up

We create a simulated environment in which isolated data is held in several nodes in order to test our proposed approaches. In each node is contained one subset of the three datasets we selected (IMBD, Yelp, Amazon). From all datasets we utilize approximately 12000 data samples (reviews) so as to simulate a realistic federated learning environment where several clients want to build a system , contributing the same amount of data. The statistics of the data that were used for our experiments are reported in Table 3.4 below.

Table 4.1: The number of reviews utilized in ours experiments from the 3 datasets

Dataset	Positive reviews	Negative reviews	Total reviews
IMDB	6500	5600	12100
Amazon	6000	6200	12200
Yelp	5700	6300	12000

We perform 5 experiments utilizing the above data:

1. In our first primary experiment, we store the 12000 data samples of each dataset separately in 3 nodes to build our federated learning model as illustrated below in Figure 3.4.

2. We repeat the first scenario but this time we store approximately 6000 data samples of each dataset separately in 3 nodes.

3. We repeat the first scenario but this time we store approximately 3000 data samples of each dataset separately in 3 nodes

4. Next, we use approximately 6000 data samples of each dataset to store separately in 6 nodes

5. Lastly, we use approximately 4000 data samples of each dataset to store separately in 9 nodes

For all of them, we pick randomly 15% of the data that is stored in each node to use as evaluation data. We perform all experiments for each neural network model that was overseen in paragraph 3.4.

The fourth and fifth experiments are carried out to find out if the same data splitted into more nodes can produce proportionate results to the first one.

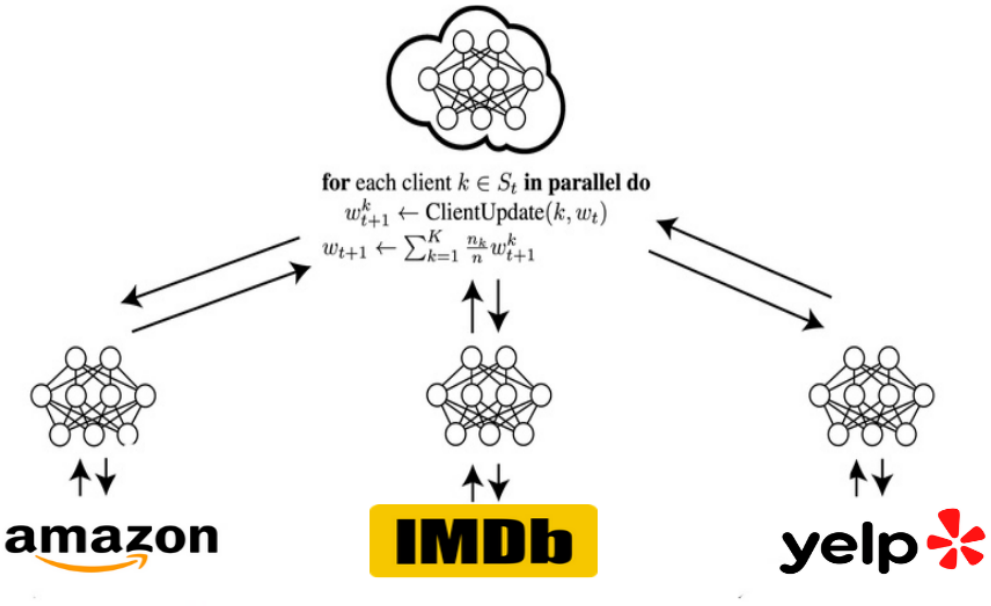


Figure 4.1: Illustration of the primary experiment of 3 clients

4.2 Evaluation Metrics

The testing method follows after the training of a deep learning model. At this point, we must evaluate our model using previously unseen data. This is accomplished by employing

several evaluation metrics that are appropriate for various types of problems. In the evaluation of our experiments we take into account only the federated (server's aggregated model) metrics - the average metrics of all batches of data trained and tested across all clients in the federated learning round. Local metrics alone cannot be taken as sign to check if our federated learning system is progressing.

To begin, we have to define the Confusion Matrix for this binary classification task. For this purpose we need to specify these four conditions:

- **True Positives (TP):** It shows the number of samples that belong to the positive class and classified correctly as positives.
- **True Negatives (TN):** It shows the number of samples that belong to the negative class and classified correctly as negatives.
- **False Positives (FP):** It shows the number of samples that belong to the negative class and classified incorrectly as positives.
- **False Negatives (FN):** It shows the number of samples that belong to the positives class and classified incorrectly as negatives.

		Actual Class	
		Positive (**P**)	Negative (**N**)
Predicted Class	Positive (**P**)	True Positive (**TP**)	False Positive (**FP**)
	Negative (**N**)	False Negative (**FN**)	True Negative (**TN**)

Figure 4.2: Confusion matrix for a binary classification problem

So for our problem we define the following metrics that are derived from the above information:

1. $Accuracy$: It represents the ratio of the correctly forecasted samples to the total number of samples. This is the most often used metric, however it is not very suitable for dealing with unbalanced data. It's calculated using the following formula:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

2. $Precision$: It expresses the ratio of the correctly forecasted positive samples to the total number of samples that were predicted as positive.

$$Precision = \frac{TP}{TP + FP}$$

3. $Recall$: It expresses the ratio of the correctly predicted positive samples to the total number of samples that are actually positive.

$$Recall = \frac{TP}{TP + FN}$$

4. $F1 - score$: It expresses the weighted average of Recall and Precision. This score is more suitable when dealing with unbalanced data because it takes both false negatives and false positives into account.

$$F1 - score = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision}$$

4.3 Results

4.3.1 Overview

In this paragraph, we present the results obtained by the deep learning models that were proposed in paragraph 3.4 for each of the cases proposed in paragraph 4.1. We discuss the perfomance of training and testing them in our federated dataset cases and draw some general conclusions. One problem that models suffered from in several cases was that of overfitting where the model works admirably on the training data, but not so well on the evaluation data. We tried to resolve this problem by using dropout layers in all model architectures which worked for most of these cases.

4.3.2 3 clients case

After hyperparameter tuning, we selected Adam as client and server optimizer for each deep learning model and ran 3 epochs of local training. For the MLP with the global average

pooling we used the values 0.001 and 0.06 as learning rates of the client and server optimizers. For the Bidirectional LSTM we used the values 0.0008 and 0.01 respectively. For the pretrained word embedding model we chose the values 0.001 and 0.003 respectively. Generally, we observed that using a smaller learning rate value for the client optimizer and a higher learning rate value for the server optimizer leads to an optimal performance.

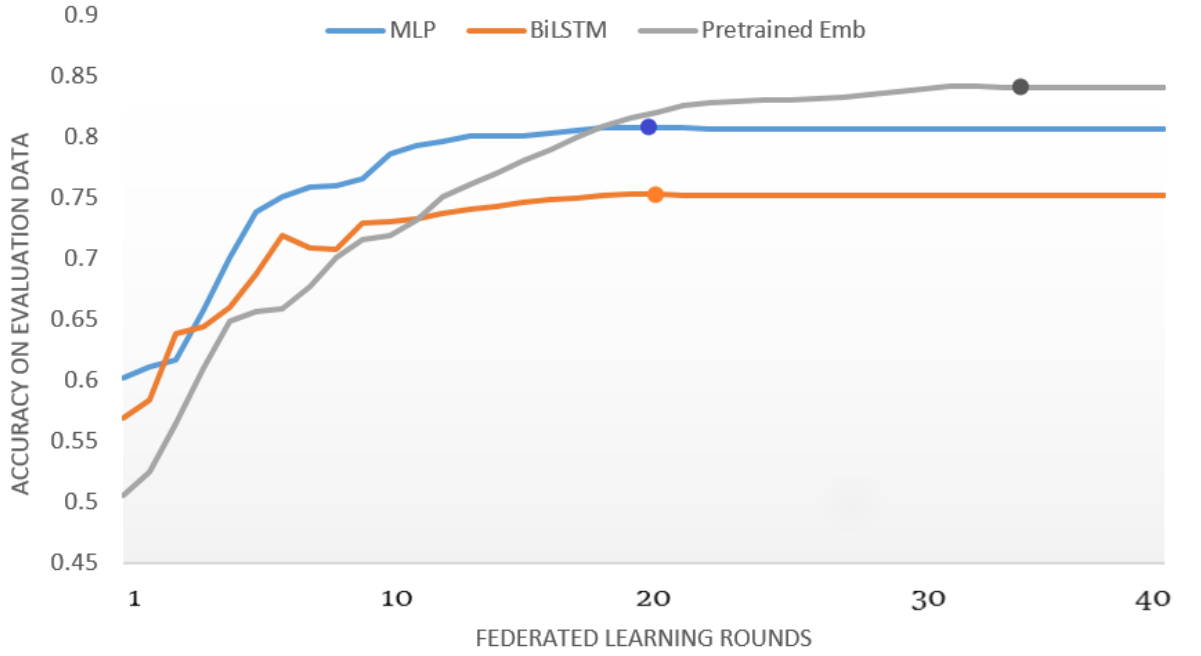


Figure 4.3: Accuracy on evaluation data for the first experiment of 3 clients

Table 4.2: Results for the first experiment of 3 clients case

Model	Accuracy	F1-score
MLP	81.2%	81.9%
Bidirectional LSTM	76.3%	76.9%
Pretrained word embedding model	83.9%	83.6%

In the 3 clients case where 12000 data samples are stored in each client the performance of our models in terms of accuracy was quite satisfactory. The first and second model (MLP and bidirectional LSTM) reached convergence after approximately 20 federated learning rounds resulting in 81.2% and 76.3% accuracy respectively. The highest accuracy of 83.9% was reached by the third model (pretrained word embedding model) after approximately 35 federated learning rounds. Nevertheless, the third model training took a much longer amount of

time because of its very large embedding size and high number of trainable parameters.

Table 4.3: Results for the second and third experiment of 3 clients case

Model	Accuracy	F1-score
MLP (second experiment)	80.6%	81.1%
MLP (third experiment)	80.2%	80.9%
Bidirectional LSTM (second experiment)	76.0%	76.5%
Bidirectional LSTM (third experiment)	76.2%	76.6%
Pretrained word embedding model (second experiment)	82.9%	83.1%
Pretrained word embedding model (third experiment)	82.2%	82.6%

By performing the second and third experiment of the 3 clients case where less data samples are stored in the 3 clients (in each client 6000 samples in the second and 3000 samples in the third), we observe that the performance of the models in terms of accuracy and f1-score drops a little in most cases, not significantly. Generally, this is something that was expected theoretically because the more data that is used to train a deep learning model , the more patterns it captures which can lead to better performance. The amount of data that is utilized to train a model has an important role in its efficiency and robustness.

4.3.3 6 clients case

After hyperparameter tuning, we selected Adam as client and server optimizer for each deep learning model and ran 3 epochs of local training. For the first model we used the values 0.005 and 0.03 as learning rates of the client and server optimizers. For the second model we used the values 0.0005 and 0.01 respectively. For the third model we chose the values 0.001 and 0.002 respectively. Generally, for each model the learning rate values of the client optimizer increased a little and the learning rate values of the server optimizer decreased a little.

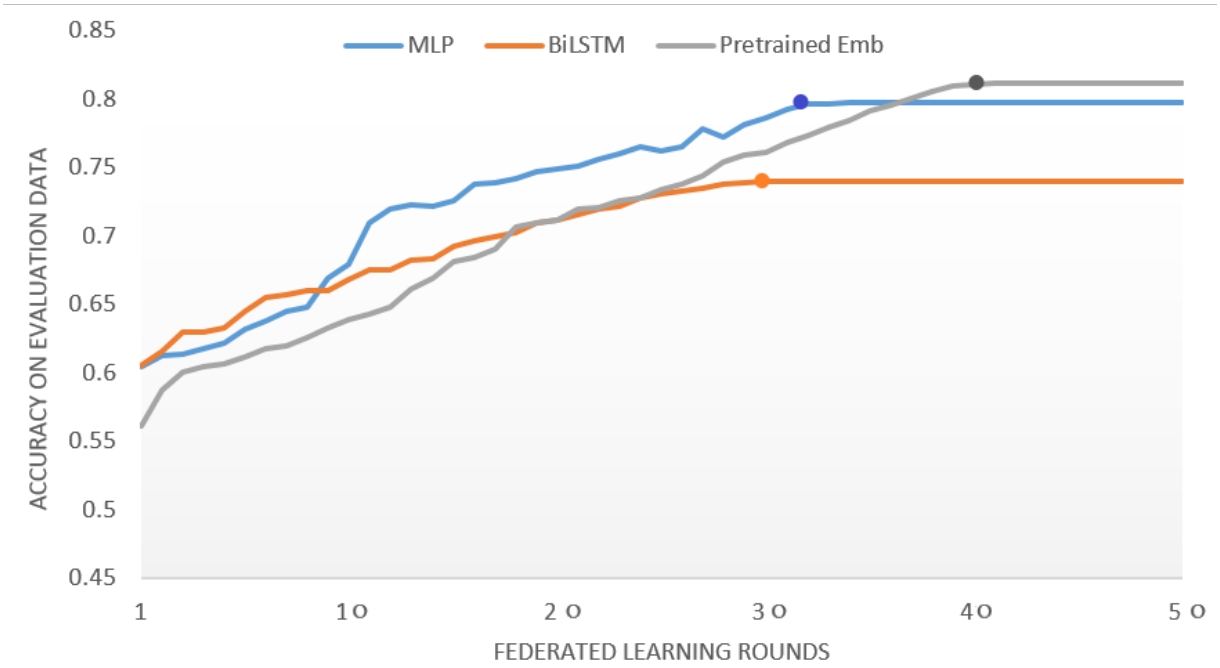


Figure 4.4: Accuracy on evaluation data for the experiment of 6 clients

Table 4.4: Results for the 6 clients case

Model	Accuracy	F1-score
MLP	80.5%	81.0%
Bidirectional LSTM	75.8%	76.1%
Pretrained word embedding model	83.2%	83.1%

In the 6 clients case the same 12000 data samples as in the 3 clients case are used but are divined into 3 more clients. So 2 clients now hold each the half of the data samples that were held by 1 node in the first case. What we expected theoretically happens in practice. The models reach almost the same performance as in the 3 nodes case after more federated learning rounds. The first and second model reach convergence after approximately 30 federated learning rounds (almost 10 more rounds than the 3 clients case) resulting in 80.5% and 75.8% accuracy respectively and the third reaches 83.2% after approximately 40 rounds (almost 10 more rounds than the 3 clients case). All models performed almost the same with a maximum difference of 0.7% from the 3 clients case performances. This is something logical as they were trained on the same data that were splitted into more clients; which also explains the need of the models for more federated learning rounds in order to converge.

4.3.4 9 clients case

After hyperparameter tuning, we selected Adam as client and server optimizer for each deep learning model and ran 3 epochs of local training. For the first model we used the values 0.008 and 0.02 as learning rates of the client and server optimizers. For the second model we used the values 0.0008 and 0.01 respectively. For the third model we chose the values 0.001 and 0.002 respectively.

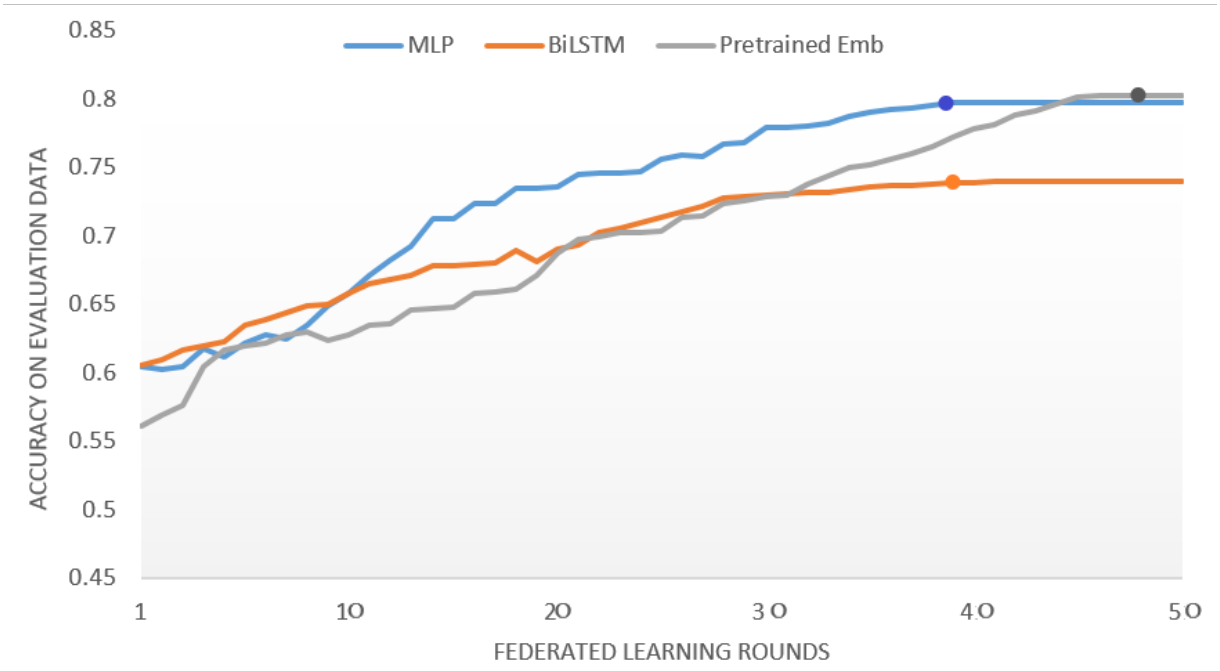


Figure 4.5: Accuracy on evaluation data for the experiment of 9 clients

Table 4.5: Results for the 9 clients case

Model	Accuracy	F1-score
MLP	80.3%	81.0%
Bidirectional LSTM	75.5%	76.0%
Pretrained word embedding model	83.0%	82.8%

In the 9 clients case the same 12000 data samples as in the 3 clients case are used but are divined into 6 more clients. So 3 clients now hold each the half of the data samples that were held by 1 client in the first case. What we expected theoretically happens in practice. The models reach almost the same performance as in the 3 clients and 6 clients after more federated learning rounds. The first and second model reach convergence after approximately

40 federated learning rounds (almost 20 more rounds than the 3 clients case and 10 more rounds than the 6 clients case) resulting in 80.2% and 75.4% accuracy respectively and the third model reaches 83.1% after approximately 48 rounds (almost 18 more rounds than the 3 clients case and 8 more rounds than the 6 clients case). All models performed almost the same with a maximum difference of 0.9% from the 3 clients case performances.

Generally, from the above experiments we can observe that as the number of clients that is utilized to store the same data increases so does the number of federated learning rounds needed to reach the same performance.

Chapter 5

Conclusions

5.1 Summary & Conclusion

In this Thesis we dealt with the task of training federated deep learning models over decentralized data for sentiment analysis on online reviews of different context. The motivation behind this was to use a privacy-preserving learning approach so that clients would not have to exchange their data in order to build together a deep learning model. For this purpose, a client-server federated learning approach was used where a central server aggregates the client model updates in order to build the federated global model. The data that were utilized came from 3 different domains: IMDB, Amazon and Yelp. Three different deep learning model architectures were utilized to test our approach: a Multilayer Perceprton with a global average pooling layer, a Bidirectional LSTM and a Pretrained word emdedding model. The results of our experiments were quite satisfactory despite the heterogeneity of data due to the fact that they come from different sources. The last model outperformed the two other ones in most cases with not significant difference. However, the pretrained model took a much longer time to be trained because of its very large embedding dimension. Generally, it was observed that the amount of data that is utilized to train a model has an important role in its efficiency and robustness. Furthermore, the results of trials were convincing enough that such approaches could work in real-world circumstances, especially when enough data is available to reinforce the model's efficiency and allow it to overcome potential barriers and constraints.

5.2 Future work

In the future, this research could be expanded by utilizing more data from different sources to store in different clients. In this way, we could implement a more realistic training loop where clients are sampled to be trained on in accordance to how much they have contributed to the training process of the global model so far. Furthermore, issues concerning the privacy preservation of the decentralized data, e.g. the homomorphic encryption of the client updates for their transfer to the server, could be explored.

Bibliography

[1] Tse Liuy Chin-Ting Changzy Chuan-Ju Wangz, Ming-Feng Tsaiy. Financial sentiment analysis for risk prediction. *Department of Computer Science Program in Digital Content and Technology National Chengchi University Taipei*, page 116, 2013.

[2] XG Li, H Li, FH Li, and H Zhu. A survey on differential privacy. *Journal of Cyber Security*, 3(5):92–104, 2018.

[3] Jibon Naher Matiur Rahman Minar. Recent advances in deep learning: An overview. 2018.

[4] Sentiment analysis definition. `https://en.wikipedia.org/wiki/Sentiment_analysis`.

[5] Sentiment analysis: Concept, analysis and applications. `https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-6c94d6f58c17`.

[6] What is artificial intelligence. `https://www.ibm.com/cloud/learn/what-is-artificial-intelligence`.

[7] Pitts W McCulloch, W.S. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[8] What are neural networks and why they matter. `https://www.sas.com/el_gr/insights/analytics/neural-networks.html#technical`.

[9] Deep neural networks. `https://www.techopedia.com/definition/32902/deep-neural-network`.

[10] Deep learning neural networks explained in plain english. https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/.

[11] Deep learning neural networks. https://www.ibm.com/cloud/learn/neural-networks.

[12] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. 12 2014.

[13] https://www.v7labs.com/blog/neural-networks-activation-functionsactivation-function. https://www.v7labs.com/blog/neural-networks-activation-functions#activation-function.

[14] Fundamentals of deep learning – activation functions and when to use them? https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/.

[15] Loss functions in machine learning. https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

[16] Common loss function and how they work. https://builtin.com/machine-learning/common-loss-functions.

[17] Various optimization algorithms for training neural network. https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

[18] An overview of gradient descent optimization algorithms. https://ruder.io/optimizing-gradient-descent/index.html#stochasticgradientdescent.

[19] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. *Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego*, 2017.

[20] Hazan E. Singer Y. Duchi, J. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[21] Dropout in (deep) machine learning. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

[22] A guide to rnn: Understanding recurrent neural networks and lstm networks. https://builtin.com/data-science/recurrent-neural-networks-and-lstm.

[23] Understanding rnn and lstm. https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e.

[24] Dan; Warnick Sean West, Jeremy; Ventura. Spring research presentation: A theoretical foundation for inductive transfer. *Brigham Young University, College of Physical and Mathematical Sciences*, 2007.

[25] Transfer learning and the art of using pre-trained models in deep learning. https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/?utm_source=blog&utm_medium=pretrained-word-embeddings-nlp.

[26] An essential guide to pretrained word embeddings for nlp practitioners. https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/.

[27] Federated learning. https://en.wikipedia.org/wiki/Federated_learning.

[28] H. B. McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *ArXiv*, abs/1602.05629, 2016.

[29] Brendan Avent Aurélien Bellet-Mehdi Bennis Arjun Nitin Bhagoji Kallista Bonawitz Zachary Charles Graham Cormode Rachel Cummings Rafael G.L. D'Oliveira Hubert Eichner Salim El Rouayheb David Evans Josh Gardner Zachary Garrett Adrià Gascón Badih Ghazi Phillip B. Gibbons Marco Gruteser Zaid Harchaoui Chaoyang He Lie He Zhouyuan Huo Ben Hutchinson Justin Hsu Martin Jaggi Tara Javidi Gauri Joshi Pe-

ter Kairouz, H. Brendan McMahan. Advances and open problems in federated learning. *Published in Foundations and Trends in Machine Learning Vol 4 Issue 1.s*.

[30] Tensorflow: An end-to-end open source machine learning platform. `https://www.tensorflow.org`.

[31] Tensorflow federated: A framework for machine learning on decentralized data. `https://www.tensorflow.org/federated`.

[32] Reza Shokri and Vitaly Shmatikov. Privacy preserving deep learning. *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, page 1310–1321.

[33] T. ryffel, federated learning with pysyft and pytorch, march 2019. `https://blog.openmined.org/upgrade-to-federated-learning-in-10-lines/`.

[34] Tensorflow encrypted. `https://github.com/tf-encrypted/tf-encrypted`.

[35] Webank ai department, federated ai technology enabler (fate). `https://github.com/FederatedAI/FATE`.

[36] Leaf: A benchmark for federated settings. `https://leaf.cmu.edu/`.

[37] comindorg: a framework for federated average settings. `https://github.com/coMindOrg/federated-averaging-tutorials`.

[38] H. Brendan McMahan and al. Communication-efficient learning of deep networks from decentralized data. 2016.

[39] Mingqing Chen, Ananda Theertha Suresh, Rajiv Mathews, Adeline Wong, Cyril Allauzen, Françoise Beaufays, and Michael Riley. Federated learning of n-gram language models, 2019.

[40] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction, 2019.

[41] Zhiqi Huang, Fenglin Liu, and Yuexian Zou. Federated learning for spoken language understanding. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3467–3478, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics.

[42] Tao Qi, Fangzhao Wu, Chuhan Wu, Yongfeng Huang, and Xing Xie. Privacy-preserving news recommendation model learning, 2020.

[43] Galina Momcheva and Plamena Tsankova. Sentiment detection with fedmd: Federated learning via model distillation. 12 2020.

[44] De-Chuan Zhan Yunfeng Shao Bingshuai Li Shaoming Song Xin-Chun Li, Lan Li. Preliminary steps towards federated sentiment classification. 2020.

[45] Yuanhe Tian Yan Song Han Qin, Guimin Chen. Improving federated learning for aspect-based sentiment analysis via topic memories. 2021.

[46] Imdb dataset of 50k movie reviews. `https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews`.

[47] Nlp datasets fast.ai. `https://course.fast.ai/datasets#nlp`.

[48] Yelp open dataset an all-purpose dataset for learning. `https://www.yelp.com/dataset`.

[49] Chun-Liang Wu and Song-Ling Shin. Machine learning based classification for sentimental analysis of imdb reviews. *Stanford University*, 2019.

[50] Pretrained text embedding nnlm-en-dim50 from tensorflow hub. `https://tfhub.dev/google/nnlm-en-dim50/2`.