Journal of Big Data

RESEARCH Open Access

Federated Freeze BERT for text classification

Omar Galal[1*], Ahmed H. Abdel-Gawad[1] and Mona Farouk[1]

*Correspondence:
omargalal@eng.cu.edu.eg

[1] Computer Engineering Department, Cairo University, Gamaa Street, Giza 12613, Egypt

Abstract

Pre-trained BERT models have demonstrated exceptional performance in the context of text classification tasks. Certain problem domains necessitate data distribution without data sharing. Federated Learning (FL) allows multiple clients to collectively train a global model by sharing learned models rather than raw data. However, the adoption of BERT, a large model, within a Federated Learning framework incurs substantial communication costs. To address this challenge, we propose a novel framework, FedFreezeBERT, for BERT-based text classification. FedFreezeBERT works by adding an aggregation architecture on top of BERT to obtain better sentence embedding for classification while freezing BERT parameters. Keeping the model parameters frozen, FedFreezeBERT reduces the communication costs by a large factor compared to other state-of-the-art methods. FedFreezeBERT is implemented in a distributed version where the aggregation architecture only is being transferred and aggregated by FL algorithms such as FedAvg or FedProx. FedFreezeBERT is also implemented in a centralized version where the data embeddings extracted by BERT are sent to the central server to train the aggregation architecture. The experiments show that FedFreezeBERT achieves new state-of-the-art performance on Arabic sentiment analysis on the ArSarcasm-v2 dataset with a 12.9% and 1.2% improvement over FedAvg/FedProx and the previous SOTA respectively. FedFreezeBERT also reduces the communication cost by 5× compared to the previous SOTA.

Keywords: Federated Learning (FL), BERT, Text classification, Pre-trained Language Models, Natural Language Processing (NLP), Sentiment analysis

Introduction

Recently, the effectiveness of Large Language Model pre-training has been demonstrated in acquiring universal language representations through extensive training on unlabeled data. Notably, Pre-trained Language Models (PLMs) like GPT [5], and BERT [6] have achieved remarkable success in various Natural Language Processing tasks, including sentiment classification [32], natural language inference [33], and question answering [16]. To fine-tune a model like BERT, a sufficient amount of data must be collected to train the whole model parameters on the downstream tasks. In the era of big data, huge data can exist but in a distributed fashion where several mobile and edge devices contain this data. Some domains such as medical domains require the data to be private and hence refuse to move their data outside their machine or organization to preserve the data privacy. To address this problem,

Federated Learning (FL) [22] proposes training several models on each client to ensure data privacy and then obtaining a global model from all the trained models as if the training is done on the whole data from all clients. To achieve this goal, clients and a main server communicate together and mainly the models are transferred from clients to the server and vice versa through several communication rounds.

Fine-tuning a Large Language Model like BERT is very challenging in a Federated Learning environment due to its huge size. BERT consists of 110 Million parameters with a total size of 651 MB. Lately, several approaches discovered BERT in Federated Learning. Hilmkil et al. [12] applied the well known FedAvg [22] on BERT, ALBERT [17] and DistilBERT [28]. Although this approach is possible, it is not very effective due to the massive communication cost needed between the server and clients. Some approaches like [19] explored freezing several BERT layers to not exchange these layers and hence reduce the communication cost. Although this strategy reduced the communication cost, the results show that the more layers you freeze, the more performance you lose on the test sets. Very recently, [20] proposed to split BERT into two parts: global and local parts. The global part consists of the first c layers of BERT and the local part is the rest of the layers. They proposed to only communicate the global part between the server and clients and hence reduce the communication cost while increasing the whole performance. Although reducing the communication cost, FedSplitBERT's experiments show that on average, the best number of layers to be included in the global part is 8; hence, the communication cost is still not reduced by a great factor compared to sending and receiving the whole model.

Being focused on text classification and to address the communication cost concern, we propose a new framework, FedFreezeBERT, referred to as **Fed**erated **Freeze BERT**. FedFreezeBERT is a novel framework for BERT-based text classification in a Federated Learning environment. Ref. [24] showed that BERT's performance can be further boosted while freezing the whole model by using an extra aggregation architecture on top of BERT such as P-SUM and H-SUM proposed by [13]. FedFreezeBERT is mainly inspired by [24] findings such that we mainly propose adding extra aggregation architecture on top of BERT and freezing the whole model parameters while only training the aggregation architecture. Following this architecture, FedFreezeBERT achieves a remarkable reduction in communication costs between the server and clients while also boosting the trained model performance. FedFreezeBERT is implemented in two versions. The first is a distributed version that we denote D-FedFreezeBERT where the learning can be done using any Federated Learning algorithm such as FedAvg to aggregate the aggregation architectures. The second is a centralized version where the embeddings extracted by BERT to the local data are sent to the server following the analogy of [9] and the server trains the aggregation architecture and then sends it to all clients.

Our contributions are summarized as follows:

- We propose FedFreezeBERT, a novel framework that combines BERT-based text classification with Federated Learning. To the best of our knowledge, FedFreezeBERT represents the most cost-effective approach that integrates BERT within a Federated Learning environment.

- Achieving a new state-of-the-art performance in Arabic sentiment classification, surpassing FedSplitBERT by a significant improvement of 1.2%.
- Reducing the communication costs with a remarkable factor of 5× compared to the previous SOTA.
- Enhancing FedSplitBERT's performance on Arabic sentiment classification with the usage of aggregation architectures, achieving an improvement of 0.66%.

The rest of the paper is structured as follows: section "Related work" shows the related work. Section "Methodology" illustrates our methodology. Section "Experiments" describes the dataset, baselines, experimental results, and further analysis. Section "Conclusion and future work" includes a conclusion to this work alongside the future work.

Related work

Federated learning

Federated Learning was first proposed by [22] where they pointed out that many modern mobile devices can have a lot of data to enrich training deep learning models. Ref. [22] proposed that the main point behind following a decentralized training approach is the privacy of the data such that many of these mobile devices may not want to share these data. They proposed Federated Learning as a general algorithm consisting of two main steps: local training on clients with their private data and server aggregation to the models generated by the mobile devices without any communication to the original data. They proposed an aggregation algorithm called FedAvg in which a weighted averaging to the parameters of the models is done and each model is weighted by its percentage of data from the total data contained by all clients. Of course, this requires all clients to have the same model architecture to be able to do the averaging. Ref. [18] proposed a generalized algorithm through reparameterization of FedAvg aggregation. They showed that FedAvg does not generalize well on non-IID - or heterogeneous—data due to the native averaging done to the model's weights. To solve this problem, Ref. [18] proposed FedProx algorithm that adds an extra regularization term to the local optimizer to force its parameters to not drift too much away from the global model. Reddi et al. [26] proposed a more general algorithm analogous to FedAvg and FedProx called FedOpt. They proposed the usage of adaptive optimizers on the server side instead of just doing simple weighted averaging. To use a general optimizer, the server needs to know not only the model parameters from each client but also the gradient. FedOpt proposed to send the gradients from clients to the server and hence any optimizer can be used on the server side.

Ref. [31] analyzed both FedAvg and FedProx and showed that there is an objective inconsistency in both of them. They proposed FedNova, a normalized averaging method that takes into account the local number of iterations done by the local solver. They showed that it solves the existing problems of FedAvg and FedProx while preserving fast error convergence. Ref. [3] proposed FedDyn, an approach that inherits the idea of FedProx of adding a regularizer. Instead of being a fixed regularizer, a dynamic regularizer was proposed by FedDyn to be dynamically updated on each client to achieve a good alignment of global and device solutions. Ref. [8] proposed FedSmart that proposes involving the local clients in the aggregation operation. FedSmart proposes packing all

updates sent to the server by clients and then sending them back to all clients. Each client has its own validation set that can be used alongside the other clients' updates to update the model locally.

Ref. [9] focused their research on how to reduce the communication cost of large Convolutional Neural Networks. The local CNN on each client is split into two parts: feature extractor and classifier. Instead of sending the model parameters, the outputs of the feature extractor are being used firstly to locally train the local classifier and secondly to be sent to the server. Not only the features but also the golden labels and the local classifier outputs are also sent to the server. At each communication round, the server collects nearly all the data from all clients in the form of features and labels and these features of course are irreversible and hence keep the data private. The server has a much larger model to be trained than compared to other clients. The server sends back its large model outputs to all clients so that the clients can use them to enhance their local small models through knowledge distillation. The idea of sending the data features instead of the model weights is very interesting and inspired us to design C-FedFreezeBERT as we send BERT's output embeddings to the server while still keeping the data secure and private.

BERT in federated learning

BERT [6] is proven to be very effective for many NLP tasks. In the context of text classification, the original paper of BERT proposed to use the [CLS] embedding while fine-tuning the model as the sentence embedding. Several approaches tried to enhance BERT's performance on text classification by proposing other methods in addition to using the [CLS] embedding. Ref. [7] proposed selecting some of the other output contextual embeddings and doing max pooling over them to get a fixed-size sentence embedding. Sentence-BERT [27] proposed fully pre-training BERT using the Siamese Network architecture [23] alongside doing extra pooling operations to the contextual embeddings like averaging or maxing. Ref. [24] proposed using different more complex aggregation architectures to improve the text classification performance such as self-attention, transformer [30], and residual connections [11]

Some research papers tried to use a large model like BERT in a Federated Learning environment. Hilmkil et al. [12] fine-tuned BERT and two of its variants ALBERT [17] and DistilBERT [28] using FedAvg. They did intensive experiments to study BERT with FedAvg on three datasets. Ref. [19] benchmarked different federated learning approaches such as FedAvg, FedProx, and FedOpt with BERT on different NLP tasks. They also proposed a framework for Federated NLP and released their code as a part of FedML framework [10]. They also studied the effect of freezing the different layers of BERT on both performance and communication costs. FedSplitBERT [20] proposed splitting BERT into two parts: local and global parts. The global part takes the first c layers of BERT as these layers capture the general language features and the local part takes the other $12 - c$ layers as these layers capture more problem-dependent features. They show that their framework is the most efficient since they don't send the whole BERT model while getting state-of-the-art results in the domain of BERT in Federated Learning. They also proposed using quantization to reduce the communication costs between the server and clients. Our approach FedFreezeBERT is mainly based on freezing BERT

using promising aggregation architectures as shown by [24] and also to improve the performance by sending the embedding in the version of C-FedFreezeBERT being inspired by [9].

Methodology

This section illustrates the details of FedFreezeBERT and its two versions. It describes the used aggregation architectures and also shows a modification proposed to FedSplitBERT [20] to enhance its performance for text classification.

FedFreezeBERT

FedFreezeBERT is inspired by [24]'s findings where BERT parameters can be frozen and an effective aggregation architecture can be trained on top of BERT to outperform BERT fine-tuning results. Since BERT's main challenge in Federated Learning is its large size, FedFreezeBERT doesn't include BERT parameters in the communication between the server and clients as the parameters don't change. We propose two versions of FedFreezeBERT: Distributed-FedFreezeBERT and Centralized-FedFreezeBERT.

Distributed-FedFreezeBERT

Distributed-FedFreezeBERT—or for short-hand D-FedFreezeBERT—is the distributed version of FedFreezeBERT where the training happens on the client side and models aggregation on the server side.

Figure 1 shows the general architecture of D-FedFreezeBERT. The server starts by initializing the weights of the pre-trained language model and then sends it to all clients. The server also initializes the weights of the aggregation architecture that is meant to be trained on the clients' sides. The server then starts the Federated Learning process by determining the number of required communication rounds T. For each communication

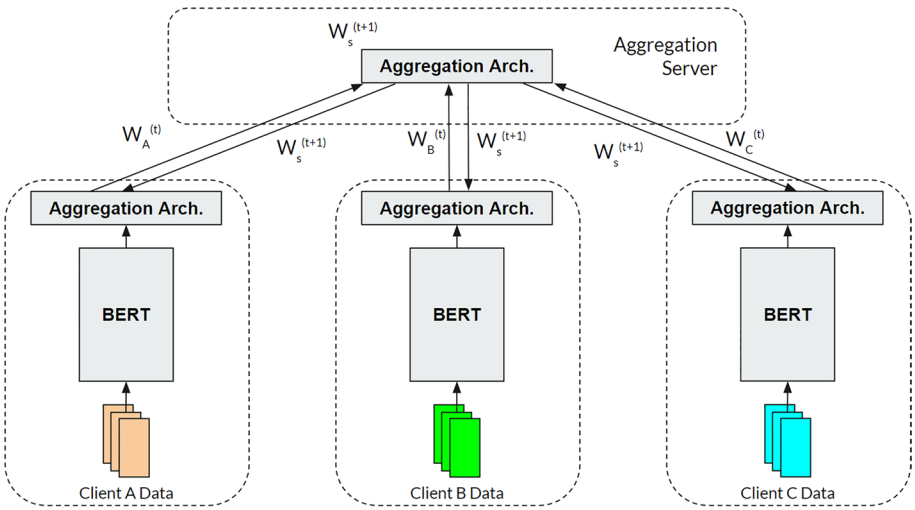

Fig. 1 The proposed architecture of D-FedFreezeBERT. All clients share the same set of parameters for BERT. In each communication round, contributing clients train locally their aggregation architecture only then send it to the server. The server aggregates these aggregation architectures with any federated learning algorithm and then sends it back to the contributing clients

round $t \in \{0, 1, 2, ..., T-1\}$, the server randomly picks a sample $S^{(t)}$ of the clients where $|S^{(t)}| \leq n_c$ and n_c is the total number of clients. The server then sends its current version of the aggregation architecture $W_s^{(t)}$ to clients in $S^{(t)}$. Each client $i \in S^{(t)}$ then replaces its local version of the aggregation architecture $W_i^{(t)}$ by the received global version of the aggregation architecture $W_s^{(t)}$. Each client uses its local data to train the parameters of the aggregation architecture to obtain $W_s^{(t+1)}$. In this step, clients can use the preferred *Optimizer*. In our case, we worked with Adam Optimizer [15]. Other optimizers can be used like SGD [14] or AdamW [35]. Each client then sends back its new version of the aggregation architecture $W_s^{(t+1)}$ to the server. After the server receives the aggregation architectures from all contributing clients in round *t*, it then aggregates them all with any known Federated Learning algorithm like FedAvg [22] or FedProx [18] to obtain the updated $W_s^{(t+1)}$. D-FedFreezeBERT is illustrated as pseudo-code in Algorithm 1.

Algorithm 1 Our proposed D-FedFreezeBERT. X_i represents the training tweets of client *i*. Y_i represents the tweets' labels of client *i*. ServerUpdate (*W*1, *W*2, ...) is a Federated Learning aggregation algorithm like FedAvg.

Server executes:
Initialize global BERT pre-trained weights and send them to all clients
for $t \in \{0, 1, 2, ..., T-1\}$ **do**
  $S^{(t)} \Leftarrow$ randomly sampled subset of clients
  for $i \in S^{(t)}$ in parallel **do**
    $W_i^{(t+1)} \Leftarrow \text{ClientUpdate}(i, W_s^{(t)})$
  end for
end for
$W_s^{(t+1)} \Leftarrow \text{ServerUpdate}(\{W_i^{(t+1)} | \forall i \in S^{(t)}\})$
send the final $W_s^{(T-1)}$ to all clients

ClientUpdate$(i, W_s^{(t)})$
$W_i^{(t)} \Leftarrow W_s^{(t)}$
Train with local data
$W_i^{(t+1)} \Leftarrow Optimizer(W_i^{(t)}, X_i, Y_i)$
return $W_i^{(t+1)}$

We can calculate the maximum amount of data transfer required for communication between clients and the server in D-FedFreezeBERT in GigaBytes as follows:

$$CommCost_{D-FedFreezeBERT} = 2 \cdot T \cdot n_c \cdot SizeOf(W_s) \quad (1)$$

Equation 1 assumes that all clients are included in all communication rounds to calculate the maximum amount of data transfer in the network. T is the number of communication rounds, n_c is the total number of clients, W_s is the aggregation architecture, and *SizeOf* is an operator that retrieves the size in GB. The setup of the whole system including sending BERT from the server to all clients in the beginning is ignored as it is a required and a standard step between all Federated Learning algorithms that use BERT.

Centralized-FedFreezeBERT

Centralized-FedFreezeBERT—or for short-hand C-FedFreezeBERT—is the centralized version of FedFreezeBERT where the training is executed on the server side. C-FedFreezeBERT is based on the fact that having BERT contextual embeddings for a sentence,

one cannot get back the original sentence and hence preserving privacy and security if these embeddings are sent to the server.

Figure 2 shows the general architecture of C-FedFreezeBERT. The server starts by sending to all clients the weights of the Pre-trained Language Model. Each client then feeds its local data through BERT to get the contextual embeddings of all tokens of all sentences which we denote Emb_i and i is the client index. All clients then send all these embeddings with the sentences' labels Y_i to the server that uses them to do the whole training without having any other communication rounds between the server and clients. After the server finishes all the epochs, it sends the final weights W_s of the aggregation architecture to all clients. C-FedFreezeBERT is illustrated as pseudo-code in Algorithm 2

Algorithm 2 Our proposed C-FedFreezeBERT. X_i and Y_i are the training data of client i. ServerTrain is a normal training loop over epochs and batches per epoch.

Server executes:

Initialize global BERT pre-trained weights and send them to all clients
for $i \in$ set of all clients in parallel **do**
    $Emb_i, Y_i \Leftarrow \text{ClientPropagate}(i)$
end for
$W_s \Leftarrow \text{ServerTrain}(\{Emb_i, Y_i | \forall i \in \{1, 2, ..., n_c\}\})$
send W_s to all clients

ClientPropagate(i)

Feed forward all sentences through BERT to get Emb_i
return Emb_i, Y_i

We can calculate the maximum amount of data transfer required for communication between clients and the server in C-FedFreezeBERT in GigaBytes as follows:

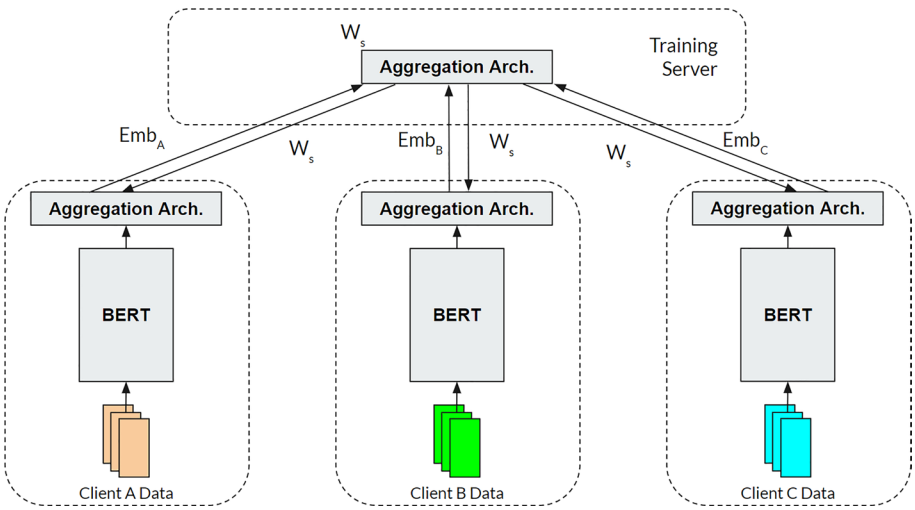

Fig. 2 The proposed architecture of C-FedFreezeBERT. All clients share the same set of parameters for BERT. Each client passes its local data through BERT to get the embeddings of these data. Clients then send these embeddings to the server that uses these embeddings to fully train the aggregation architecture. After the training is done, The server sends the final weights of the aggregation architecture back to all clients

$$CommCost_{C-FedFreezeBERT} = \frac{4}{2^{30}} \cdot n_s \cdot n_t \cdot dim_{emb} + n_c \cdot SizeOf(W_s) \quad (2)$$

Where n_s is the total number of sentences over all clients, n_t is the maximum sequence length used by BERT which is 128 in our case, dim_{emb} is BERT embedding dimension and it is 768 in our case. The fraction $\frac{4}{2^{30}}$ is because each number in the embedding is represented by 4 bytes and the total term is divided by 2^{30} to get the size of all the embeddings in GigaBytes. n_c is the total number of clients, W_s is the aggregation architecture, and *SizeOf* is an operator that retrieves the size in GB. The setup of the whole system including sending BERT from the server to all the clients in the beginning is ignored as it is a required and a standard step between all Federated Learning algorithms that use BERT.

BERT aggregation architectures

The purpose of the aggregation architectures is to aggregate BERT's final contextual embeddings and its hidden layers' embeddings to get a more representative sentence embedding in the context of text classification. Omar et al. [24] shows different architectures to aggregate BERT's embedding for text classification. They also show that some aggregation architectures can be used with BERT parameters kept frozen to outperform BERT being fine-tuned. Our research will use four aggregation architectures with BERT for text classification. The first two architectures are simple and aggregate BERT's final layer embeddings. The last two architectures are more complex as they aggregate BERT's final and hidden layers' embeddings. The architectures we use can be summarized as follows:

- Ordinaryaggregator: This architecture is the common and standard way where BERT's [CLS] output embedding is used as the sentence embedding. The [CLS] embedding is then fed to the classifier which is a simple linear layer in our case.
- Average aggregator: This aggregation architecture is an intuitive and very simple one in which BERT's all final layer contextual embeddings are averaged to get a fixed size embedding representing the input sentence. Ref. [24] shows that although this aggregation architecture is very simple, it can achieve high performance when BERT parameters are kept frozen. The authors also show that its results are very near to using [CLS] embedding with fine-tuning BERT. We decided to include this aggregation architecture in our experiments as it is very simple and yet effective when BERT is frozen.
- P-SUM: This architecture is first proposed by [13] to improve BERT performance for aspect-based sentiment analysis. Ref. [24] then proposed that this architecture performance can be further improved if BERT parameters are kept frozen. The architectural details are shown in Fig. 3. An extra four BERT layers are added on top of BERT in parallel. The four final BERT layers pass their outputs to the extra BERT layers in parallel. Each path from the four parallel paths then acts as a classifier and in the training, the four classifiers' losses are added together. In inference time, the four classifiers' outputs are averaged to get the final predictions. As shown by [13] and [24], the best number of last layers to be used is four and hence we decided to follow this number in the experiments.

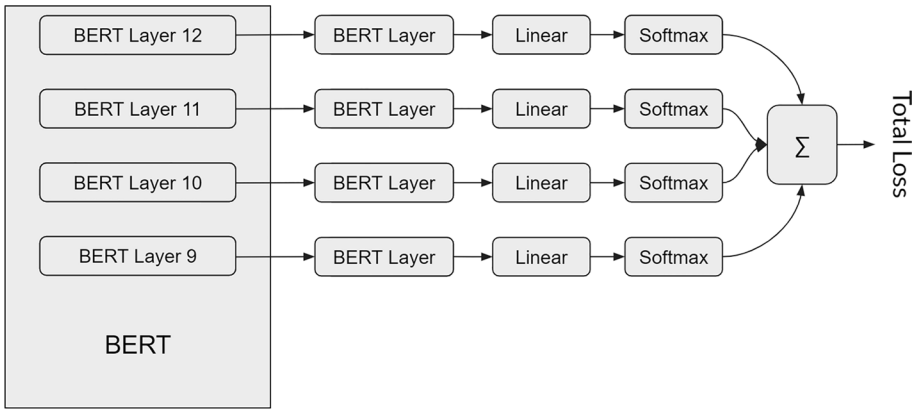

Fig. 3 P-SUM aggregation architecture [24]

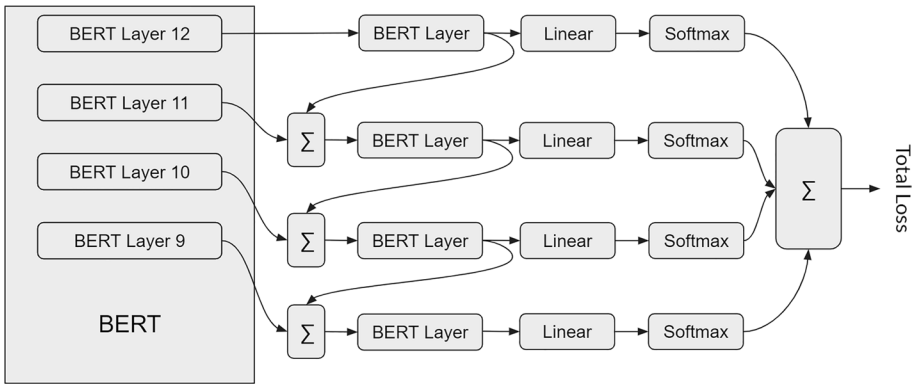

Fig. 4 H-SUM aggregation architecture [24]

- H-SUM: This architecture is first proposed by [13] to improve BERT performance for aspect-based sentiment analysis. Ref. [24] then proposed that this architecture performance can be further improved if BERT parameters are kept frozen. The architectural details are shown in Fig. 4. Extra four BERT layers are added on top of BERT in a hierarchical fashion such that each extra BERT layer adds its outputs to the input of the extra BERT layer that precedes it. Each path from the four parallel paths then acts as a classifier and in the training, the four classifiers' losses are added together. In inference time, the four classifiers' outputs are averaged to get the final predictions. As shown by [13] and [24], the best number of last layers to be used is four and hence we decided to follow this number in the experiments.

Modified FedSplitBERT

FedSplitBERT [20] is a Federated Learning framework to fine-tune BERT for general NLP tasks. It is based on splitting BERT's 12 layers into two parts: a global part and a local part. The global part is the first c layers and the local part is the last $12 - c$ layers. For the local training on clients in each communication round, both global and local

layers parameters change based on the local data. The global part only is sent from clients to the server for aggregation and then the server sends the aggregated global part back to all clients at the end of each communication round. For text classification, FedSplitBERT is based on the following settings:

1. BERT is fine-tuned meaning that all the parameters are changed during the training.
2. BERT's [CLS] embedding is used as the sentence embedding to be fed to the classifier.
3. The global part is aggregated on the server with FedAvg.

Our modification to FedSplitBERT covers the second point. We propose using AverageAggregator, P-SUM, and H-SUM to improve FedSplitBERT performance for text classification. The first point of course couldn't be changed because if we don't fine-tune the model, there is no meaning to split the model into global and local parts and then aggregate only the global part to the server. For the third point, it is already mentioned by FedSplitBERT authors that other Federated Learning algorithms other than FedAvg can be used but we decided to follow exactly their approach to be consistent with their results. FedSplitBERT's maximum communication cost can be calculated as follows:

$$CommCost_{FedSplitBERT} = 2 \cdot T \cdot n_c \cdot (c \cdot SizeOf(BL) + SizeOf(BE)) \tag{3}$$

Where T is the number of communication rounds, n_c is the total number of clients, and c is the critical layer defined by FedSplitBERT or in other words the number of BERT layers in the global part. *BL* is an abbreviation of BERT Layer. *BE* is an abbreviation of BERT's Embedding Layer. *SizeOf*(.) is an operator that returns the size of an object in GigaBytes.

Experiments

Dataset and evaluation metric

The dataset used in this work is ArSarcasm-v2 published by [2]. The dataset is a benchmark for Arabic sentiment analysis and sarcasm detection. The dataset consists of tweets collected through Twitter API [21] and manually labeled for sentiment and sarcasm. All tweets are Arabic in language but have different dialects and hence are more challenging for sentiment and sarcasm. This work mainly addresses the problem of sentiment analysis through ArSarcasm-v2. The dataset is split into 12k tweets for training and 3k tweets for testing. Table 1 shows the sentiment distribution over training tweets of ArSarcasm-v2.

Table 1 ArSarcasm-v2 sentiment distribution over training tweets

Task	Class	Count
Sentiment	Positive	2180
	Negative	4621
	Neutral	5747
Total		12,548

To evaluate our experiments on ArSarcasm-v2, the same evaluation metric F_1^{PN} defined in [2] and also followed in [24] is also followed in this work. F_1^{PN} is the average F_1-score over the positive and negative classes and can be formulated as follows:

$$F_1^{PN} = \frac{F_1^{positive} + F_1^{negative}}{2} \tag{4}$$

Pre-trained language model

Working with Arabic sentences, there is a variety of Pre-trained Language Models with BERT-base [6] architecture. Following the work done by [1, 2], and [24], we decided to use MARBERT [1] since it is the most effective PLM when working with the Arabic language, especially with Arabic tweets. MARBERT is pre-trained on 128 GB of text with about 1 Billion tweets. The pre-trained weights of the model were downloaded from Hugging Face [34].

Baseline methods

This work considers three main Federated Learning baseline methods: FedAvg [22], FedProx [18], and FedSplitBERT [20] in addition to the central training approach. The reason behind considering both FedAvg and FedProx is that nearly all new Federated Learning approaches compare their results to both FedAvg and FedProx as they are the main and fundamental algorithms in Federated Learning. The implementation of FedAvg and FedProx was used from FedML [10]. The communication cost for both FedAvg and FedProx with BERT can be calculated as follows:

$$CommCost_{FedAvg} = 2 \cdot T \cdot n_c \cdot SizeOf(BERT) \tag{5}$$

Where T is the total number of communication rounds, n_c is the total number of clients, and $SizeOf(BERT)$ is the total size in GB of the original BERT model for sequence classification.

For FedSplitBERT, we couldn't find an official implementation to it hence we implemented our version with the same architecture specified by FedSplitBERT. To make sure that our implementation is correct and matches the original FedSplitBERT's implementation, We evaluated our implementation on SST-2 [29] with the critical layer $c = 8$ and the same settings defined by [20]. Table 2 shows the performance of the original FedSplitBERT with the accuracy of 93.27% reported from their original paper. The table shows that our implementation gives almost identical accuracy on SST-2 and hence we can continue experimenting with our own implementation.

Table 2 Performance of original and our implementation of FedSplitBERT on SST-2

FedSplitBERT	Accuracy
Original [20] (from paper)	93.27
Our implementation	93.29

Experimental setup

Implementation

All the architectures are implemented in Pytorch [25]. All Federated Learning simulations are done with FedML [10] single process simulation. The experiments were executed on Google Colab [4] on Nvidia Tesla T4. MARBERT maximum sequence length was set to 128 following both [2] and [24]. The training data was distributed among 5 clients as described in [19]. The loss function to train the models is chosen to be the Cross-Entropy Loss. The network parameters were optimized with Adam Optimizer [15]. For D-FedFreezeBERT, the server can use any Federated Learning aggregation algorithm and we decided to do the experiments with both FedAvg and FedProx.

Hyperparameters

Training was done with the batch size set to 16. Since we have two main categories of training: distributed and centralized, we wanted the training process to be as fair as possible. Distributed training approaches such as FedAvg, FedProx, FedSplitBERT, and D-FedFreezeBERT require the training to be done on the client side and having multiple communication rounds between the server and clients. Centralized approaches such as C-FedFreezeBERT have no communication rounds and the full training is done on the server side. The maximum number of epochs suggested by [2] and [24] was 5 and hence we executed C-FedFreezeBERT experiments with 5 epochs and other approaches with 5 communication rounds while each client executes 1 epoch per round. Following this way, we guarantee that all approaches see each training tweet the same number of times and hence the results can be fairly compared. FedProx λ was tuned and finally set to 0.05. FedSplitBERT critical layer c is used to be 8 as it gives the best performance as observed by [20]. Learning rates in the range from $9e^{-4}$ to $1e^{-6}$ were explored. The final learning rates used are: $1e^{-6}$ with FedAvg and FedProx, $1e^{-5}$ with FedSplitBERT, and $5e^{-4}$ with D-FedFreezeBERT and C-FedFreezeBERT.

Results and analysis

Table 3 shows the F_1^{PN} metric for baseline methods and our proposed approaches on the ArSarcasm-v2 testset. All baseline approaches use the OrdinaryAggregator where the [CLS] embedding is used as the sentence embedding. We also experimented with FedFreezeBERT with the [CLS] embedding although it is expected to get low performance. It is commonly known that it is very challenging for Federated Learning algorithms to get close or higher performance compared to the central approaches and hence we included central training to compare the different Federated Learning algorithms and frameworks to it. Table 3 shows that both FedAvg and FedProx get low performance compared to the central training approach. D-FedFreezeBERT gets very low F_1^{PN} which is expected because freezing BERT is known to degrade the performance with the [CLS] embedding being used as a sentence embedding. C-FedFreezeBERT has also a low performance compared to other algorithms while being used with the OrdinaryAggregator but its performance is much higher than

Table 3 Comparison of FedFreezeBERT with other baseline approaches on ArSarcasm-v2 test set

Baseline Methods and FedFreezeBERT with OrdinaryAggregator		
Method	**BERT Frozen?**	F_1^{PN}
Central Training	No	73.48
FedAvg	No	66.67
FedProx	No	66.36
FedSplitBERT	No	**74.39**
D-FedFreezeBERT (FedAvg)	Yes	49.39
D-FedFreezeBERT (FedProx)	Yes	47.5
C-FedFreezeBERT	Yes	65.1

F_1^{PN} **Using Advanced Aggregation Architectures**				
Method	**BERT Frozen?**	**AverageAggregator**	**P-SUM**	**H-SUM**
Central Training	No	73.00	74.63	73.93
FedAvg	No	63.84	71.71	68.81
FedProx	No	63.55	71.56	68.54
FedSplitBERT	No	**74.08**	74.57	74.88
D-FedFreezeBERT (FedAvg)	Yes	54.23	74.18	74.29
D-FedFreezeBERT (FedProx)	Yes	52.09	74.13	74.21
C-FedFreezeBERT	Yes	72.34	**75.26**	**74.94**

Bold values indicate the best-performing approach when the aggregation architecture is fixed

D-FedFreezeBERT because the whole training happens in one place and hence no heterogeneity in the data exists. FedSplitBERT is the best baseline approach outperforming the central training approach

When AverageAggregator is used, the central training performance degraded a little. We expect that both FedAvg and FedProx patterns follow the central approach and hence their performance also degraded with AverageAggregator compared to OrdinaryAggregator. Also, FedSplitBERT performance degraded a little and these observations follow the findings of [24]. Both D-FedFreezeBERT and C-FedFreezeBERT performance was enhanced because of using the AverageAggregator. Although using the AverageAggregator improved FedFreezeBERT, it is not a very powerful aggregation architecture but it is very efficient because of its simplicity.

P-SUM is proven to be a powerful aggregation architecture for text classification by [13] and [24]. The central training performance improved to achieve 74.63 F_1^{PN}. Both FedAvg and FedProx performance also improved compared to their counterparts with the OrdinaryAggregator but as expected they cannot exceed the central training performance. For FedSplitBERT, the performance was enhanced with P-SUM by a little margin but its performance couldn't exceed the central training performance. D-FedFreezeBERT achieved a comparably high F_1^{PN} which is very interesting. C-FedFreezeBERT with P-SUM achieved 75.26 F_1^{PN} which is a new state-of-the-art on ArSarcasm-v2 in a Federated Learning setting. It is worth noting that C-FedFreezeBERT with P-SUM achieved an improvement of 12.9%, 2.4%, and 1.2% over FedAvg/FedProx, baseline central training, and FedSplitBERT respectively.

When experimenting with the different approaches with H-SUM, it can be shown that central training, FedAvg, and FedProx get a performance enhancement compared to their baselines counterparts. When they are compared with the P-SUM results, it

can be observed that their performance degrades and this is because H-SUM has a long chain of 14 BERT layers to be fine-tuned if the model parameters aren't frozen and this makes the learning process more complex. FedSplitBERT with H-SUM is improved by 0.66% over the baseline FedSplitBERT which shows that aggregation architectures can help improve existing promising approaches. To make sure that the 0.66% is a relevant improvement, the standard deviation of the original FedSplitBERT and FedSplitBERT with H-SUM are estimated over five runs and we found them to be 0.12% and 0.17% respectively. D-FedFreezeBERT's performance increased a little with H-SUM compared to its performance with P-SUM. C-FedFreezeBERT with H-SUM achieved 74.94 F_1^{PN} exceeding also all other approaches - except for C-FedFreezeBERT with P-SUM - with an improvement of 12.4%, 2%, and 0.74% over FedAvg/FedProx, baseline central training, and FedSplitBERT respectively.

To measure the communication cost and hence communication gain, which is defined as the ratio of the baseline's communication cost to the optimized approach's communication cost, of the different methods, we measured in the conducted experiments all sent and received data between the server and clients and verified their correctness with equations 1, 2, 3, and 5. The parameters used in our experiments are as follows:

- $n_c = 5$
- $T = 5$
- $n_s = 12548$
- $n_t = 128$
- $c = 8$
- $SizeOf\,(W_s) = 113.44MB$
- $dim_{emb} = 768$
- $SizeOf\,(BL) = 28.35MB$
- $SizeOf\,(BE) = 308MB$
- $SizeOf\,(BERT) = 651.37MB$

Table 4 shows the communication cost and communication gain of the different Federated Learning methods under the setting of our experiments on the ArSarcasm-v2 dataset. The table shows the effectiveness of both D-FedFreezeBERT and C-FedFreezeBERT. D-FedFreezeBERT achieves a communication gain of 5.7× and 4.7 × compared to FedAvg and FedSplitBERT respectively. C-FedFreezeBERT achieves a communication gain of 6.2× and 5× compared to FedAvg and FedSplitBERT respectively. It is worth noting

Table 4 Comparison of different approaches communication cost and communication gain on ArSarcasm-v2

Method	Communication cost (GB)	Communication gain
FedAvg/FedProx	31.8	1×
FedSplitBERT	26.11	1.2×
D-FedFreezeBERT (P-SUM/HSUM)	5.54	5.7×
C-FedFreezeBERT	5.15	6.2×

that FedSplitBERT achieved state-of-the-art results not only on a performance metric such as F_1^{PN} but also from the communication cost perspective.

Conclusion and future work

This work proposes a novel framework, FedFreezeBERT, for BERT-based text classification in Federated Learning environments. FedFreezeBERT mainly addresses the challenge of the expensive communication cost of BERT in Federated Learning by freezing the model parameters and boosting its performance with aggregation architectures such as P-SUM and H-SUM. FedFreezeBERT is evaluated on Arabic sentiment analysis on the ArSarcasm-v2 dataset. The experiments show that FedFreezeBERT's performance in sentiment analysis exceeds FedAvg and FedProx performance with a 12.9% improvement and exceeds FedSplitBERT performance with a 1.2% improvement. The experiments also show that FedFreezeBERT is the most cost-effective framework regarding communication costs with a communication gain of 6.2× and 5× compared to FedAvg/FedProx and FedSplitBERT respectively. FedFreezeBERT is a simple and easy-to-implement approach. Also, D-FedFreezeBERT is compatible with nearly all Federated Learning aggregation algorithms such as FedAvg and FedProx. In the future, FedFreezeBERT can be explored outside the context of text classification in problems such as question answering, semantic similarity, etc.

Acknowledgements
Not applicable.

Author contributions
MF provided the initial idea for the research. OG researched the problem to identify the research gap. OG proposed FedFreezeBERT and the extra enhancements for FedSplitBERT. OG implemented the experiments and analyzed the results. OG wrote the manuscript. MF did the manuscript revision. Both MF and AA supervised OG during the research.

Funding
Open access funding provided by The Science, Technology & Innovation Funding Authority (STDF) in cooperation with The Egyptian Knowledge Bank (EKB).

Availability of data and materials
All data generated or analyzed during this study are included in this published article. Derived data supporting the findings of this study are available from the corresponding author on request.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Received: 31 August 2023 Accepted: 19 January 2024
Published online: 09 February 2024

References

1. Abdul-Mageed M, Elmadany A, Nagoudi EMB. Arbert & marbert: deep bidirectional transformers for arabic. arXiv preprint. 2020. arXiv:2101.01785.
2. Abu Farha I, Zaghouani W, Magdy W (2021) Overview of the WANLP 2021 shared task on sarcasm and sentiment detection in Arabic. In: Proceedings of the Sixth Arabic Natural Language Processing Workshop. Association for Computational Linguistics, Kyiv, Ukraine (Virtual). 2021. pp. 296–305, https://aclanthology.org/2021.wanlp-1.36.

3. Acar DAE, Zhao Y, Navarro RM, et al. Federated learning based on dynamic regularization. arXiv preprint. 2021. arXiv:2111.04263.
4. Bisong E, Bisong E. Google Collaboratory. Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners. 2019. pp. 59–64.
5. Brown T, Mann B, Ryder N, et al. Language models are few-shot learners. Adv Neural Inform Process Syst. 2020;33:1877–901.
6. Devlin J, Chang MW, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018. arXiv preprint arXiv:1810.04805.
7. Gao Z, Feng A, Song X, et al. Target-dependent sentiment classification with bert. IEEE Access. 2019;7:154290–9.
8. He A, Wang J, Huang Z, et al. Fedsmart: An auto updating federated learning optimization mechanism. In: Web and Big Data: 4th International Joint Conference, APWeb-WAIM 2020, Tianjin, China, September 18-20, 2020, Proceedings, Part I 4. Cham: Springer; 2020. pp. 716–724.
9. He C, Annavaram M, Avestimehr S. Group knowledge transfer: federated learning of large cnns at the edge. Adv Neural Inform Process Syst. 2020;33:14068–80.
10. He C, Li S, So J, et al. Fedml: A research library and benchmark for federated machine learning. arXiv preprint. 2020. arXiv:2007.13518.
11. He K, Zhang X, Ren S, et al (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. pp. 770–778.
12. Hilmkil A, Callh S, Barbieri M, et al (2021) Scaling federated learning for fine-tuning of large language models. In: Natural Language Processing and Information Systems: 26th International Conference on Applications of Natural Language to Information Systems, NLDB 2021, Saarbrücken, Germany, June 23–25, 2021, Proceedings, Springer; 2021. pp. 15–23.
13. Karimi A, Rossi L, Prati A. Improving bert performance for aspect-based sentiment analysis. arXiv preprint. 2020. arXiv:2010.11731.
14. Ketkar N, Ketkar N. Stochastic gradient descent. In: Ketkar N, editor. Deep learning with python: a hands-on introduction. Berkeley: Apress; 2017. p. 113–32.
15. Kingma DP, Ba J. Adam: a method for stochastic optimization. arXiv preprint. 2014. arXiv:1412.6980.
16. Lai G, Xie Q, Liu H, et al. Race: Large-scale reading comprehension dataset from examinations. arXiv preprint. 2017. arXiv:1704.04683.
17. Lan Z, Chen M, Goodman S, et al. Albert: A lite bert for self-supervised learning of language representations. arXiv preprint. 2019. arXiv:1909.11942.
18. Li T, Sahu AK, Zaheer M, et al. Federated optimization in heterogeneous networks. Proc Mach Learn Syst. 2020;2:429–50.
19. Lin BY, He C, Zeng Z, et al. Fednlp: a research platform for federated learning in natural language processing. arXiv preprint. 2021. arXiv:2104.08815.
20. Lit Z, Sit S, Wang J, et al. Federated split bert for heterogeneous text classification. In: 2022 International joint conference on neural networks (IJCNN), IEEE; 2022. pp 1–8.
21. Makice K. Twitter API: up and running: learn how to build applications with the Twitter API. Sebastopol: O'Reilly Media Inc; 2009.
22. McMahan B, Moore E, Ramage D, et al. Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics, PMLR; 2017. pp. 1273–1282.
23. Melekhov I, Kannala J, Rahtu E. Siamese network features for image matching. In: 2016 23rd international conference on pattern recognition (ICPR), IEEE; 2016. pp. 378–383
24. Galal O, Abdel-Gawad AH, Farouk M. Rethinking of bert sentence embedding for text classification. Research Square preprint. 2024. https://doi.org/10.21203/rs.3.rs-3920665/v1.
25. Paszke A, Gross S, Chintala S, et al. Pytorch Computer software Version. 2016;03:1.
26. Reddi S, Charles Z, Zaheer M, et al. Adaptive federated optimization. arXiv preprint. 2020.arXiv:2003.00295.
27. Reimers N, Gurevych I. Sentence-bert: Sentence embeddings using siamese bert-networks. 2019. arXiv preprint arXiv:1908.10084.
28. Sanh V, Debut L, Chaumond J, et al. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv preprint. arXiv:1910.01108.
29. Socher R, Perelygin A, Wu J, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 conference on empirical methods in natural language processing. 2013. pp. 1631–1642.
30. Vaswani A, Shazeer N, Parmar N, et al (2017) Attention is all you need. Adv Neural Inform Process Syst. 2017; 30.
31. Wang J, Liu Q, Liang H, et al. Tackling the objective inconsistency problem in heterogeneous federated optimization. In: Larochelle H, Ranzato M, Hadsell R, et al., editors. Advances in neural information processing systems, vol. 33. Red Hook: Curran Associates Inc; 2020. p. 7611–23.
32. Wang Y, Huang M, Zhu X, et al. Attention-based lstm for aspect-level sentiment classification. In: Proceedings of the 2016 conference on empirical methods in natural language processing. 2016. pp. 606–615.
33. Williams A, Nangia N, Bowman SR. A broad-coverage challenge corpus for sentence understanding through inference. arXiv preprint. 2017. arXiv:1704.05426.
34. Wolf T, Debut L, Sanh V, et al. Transformers: state-of-the-art natural language processing. In: Liu Q, Schlangen D, editors., et al., Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations. Stroudsburg: Association for Computational Linguistics; 2020. p. 38–45.
35. Zhuang Z, Liu M, Cutkosky A, et al. Understanding adamw through proximal methods and scale-freeness. arXiv preprint. 2022. arXiv:2202.00089.

Publisher's Note