Homomorphic learning: A privacy-focused approach to deep learning

Aadesh Neupane[1]

I. INTRODUCTION

Deep Neural Networks (DNNs) has been helping humans to tackle severe problems, ranging from weather forecasting [1] to human trafficking [2], in turn saving millions of lives and dollars. Still, many sensitive areas such as health and government application are yet to be revolutionized using deep learning techniques. The most crucial and beneficial application of artificial intelligence algorithms requires the use of very delicate and private information of a user. A recent data breach of claims data from Cigna [3] and location history from the Strava [4] had created a buzz around the AI researchers. The widespread account of Apple, Google, and Amazon employees listening to our request via Siri, Assistant, and Alexa are alarming [5]. It forces us to wonder if any of our personal information is safe on our phone, computer, and cloud.

With the concerns about user's data lately, there has been a community-wide effort to learn from data without access to the user's private data in raw form. Homomorphic learning is one way to learn machine learning models without access to raw user's data. The user's data is encrypted using a private key, and the learning algorithm only has access to encrypted data. The prediction/output from the ML models is sent to the user. Only the user has the capability of decrypting and interpreting the results. Thus this method can be used to process critical information without ever worrying about the data breach and privacy issues.

The most famous example for the use of homomorphic learning is where a data owner wants to send data up to the cloud for processing but does not trust a service provider with their critical data. Using a homomorphic encryption scheme, the data owner encrypts their data and sends it to the server. The server performs the relevant computations on the data without ever decrypting it and sends the encrypted results to the data owner. The data owner is the only one able to decrypt the results since they alone have the secret key. Figure 1 show a simple flow in homomorphic learning. The critical component of this learning mechanism is homomorphic encryption, and a brief introduction is provided in the following paragraph.

Fully Homomorphic Encryption (FHE) makes it possible to perform arbitrary computations (mathematical operations like sum or product as well as more complicated operations) on encrypted data and without needing a secret key, and while keeping the individual pieces of data hidden and secure [6]. For example, a web application could prepare an individual's tax return using her encrypted financial information, without seeing any data in the raw form.

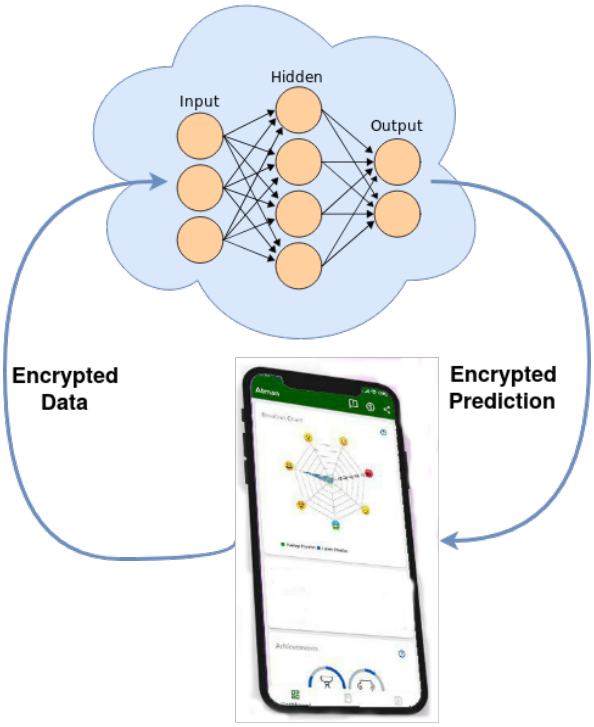

Fig. 1. A simple system processing the user's sensitive information using homomorphic learning.

For the class project, we propose to implement a language model for sentiment analysis on encrypted data using homomorphic encryption and deep learning. First, we try HeLib and SEAL cryptography library to encrypt the MNIST image data. With strong inference with the trained model on MNIST image data, we will explore different architectures for sentiment analysis. Finally, we will try to train a language model with encrypted data.

II. RELATED WORK

The publication of Gentry's doctorate dissertation [6] in 2009 made the fully homomorphic encryption (FHE) a buzz word. Still, FHE was not computationally efficient to be applied to real-world applications. Over the last few years, with extensive research and opensource tools like *Seal* [7], *Palisade* [8], *HElib* [9] and others started to change the landscape of the homomorphic encryption. The integration of these tools into popular Machine Learning libraries such as TensorFlow and PyTorch enabled amateurs and researches

[1]Aadesh Neupane is graduate student, Brigham Young University aadeshnpn@byu.edu

alike to build applications focused on privacy. These open-source tools are responsible for the homomorphic encryption to reach the masses. Still, only a few specialized companies provide homomorphic learning as a service to healthcare and government organizations, and it might take a few more years for homomorphic learning to be ubiquitous among the cloud service providers.

Currently, there are a few frameworks like PySyft [10] and HE Transformer [11] that allow the users to perform homomorphic learning with Neural Network models on encrypted data. These frameworks decouple private data from model training using Federated Learning, Differential Privacy, and Multi-Party computation (MPC) within the leading deep learning frameworks like PyTorch and TensorFlow.

Dowlin et al. [12] were the first to train a neural network model on encrypted MNIST data with high throughput and accuracy using the SEAL [7] library for homomorphic encryption. Hesamifard et al. [13] trained a deep neural network over encrypted MNIST and CIFAR data with high efficiency and accuracy. Badawi [14] et al. trained the first homomorphic convolution neural network (HCNN) on encrypted data with Graphics Processing Unit (GPU)s. All previous approaches before HCNN ran only on the CPU. The HCNN is truly remarkable, as it achieved the high-security level and high classification accuracy for MNIST and CIFAR. The famous MNIST image dataset was encrypted using the Crypten library so that the shape of a tensor is preserved.

Lou et al. [15] developed an FHE-based scheme called *Glyph* to fast and accurately train DNNs on encrypted data by switching between TFHE (Fast Fully Homomorphic Encryption over the Torus) and BGV cryptosystems. This framework uses logic-operation-friendly TFHE to implement non-linear activations, while adopts vectorial-arithmetic-friendly BGV to perform multiply-accumulation (MAC) operations. It further applies transfer learning on the training of DNNs to improve the test accuracy and reduce the number of MAC operations between ciphertext and ciphertext in convolutional layers. The framework was able to achieve 98.6% accuracy on the MNIST dataset.

III. Approach

We explored different homomorphic encryption libraries such as Seal, HeLib, Intel's HE Transformer, PySyft, and FB's crypto. After the initial assessment, we found a hackish way to use FB's crypto library for homomorphic learning. Currently, FB's crypto library does not natively support homomorphic encryption but found an internal hack to use the crypto tensors. We follow the process below for homomorphic learning:

- User load their normal dataset and convert the data into PyTorch tensor
- Convert the PyTorch tensor to crypo tensor using Crypten library
- Extract the encrypted tensor and save it to database
- Transfer the encrypted tensor to server for computation
- Server performs computation on the encrypted data and sends back the encrypted predictions to the user

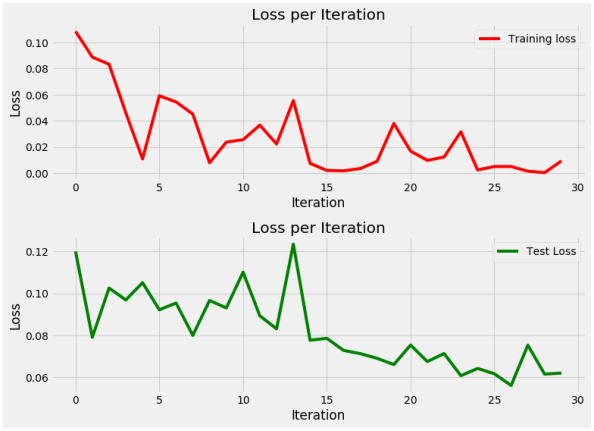

Fig. 2. The performance of CNN on encrypted MNIST

- User decryption the predictions

IV. Experiments

A. Encrypted MNIST

The famous MNIST image dataset [16] was encrypted using Crypten [17] library so that the shape of a tensor is preserved. The original dataset images values were between 0 to 1 with normalization applied. When encrypting the images with the Crypten library, the floating-point values are converted to an integer value keeping the shape of the images intact. Also, the labels are encrypted such that the labels are higher random integer values.

The encrypted MNIST dataset is loaded and normalized as the *cudadnn* library did not support long integer values for convolution, at least in our environment setup. We trained a DNN with two convolution layers and two fully connected layers with the last layer with ten nodes. The network trained on the encrypted dataset was able to achieve 99% accuracy on the test set. Figure 2 shows the performance of the CNN model on the encrypted dataset.

The trained model is stored on the server. As described earlier, the goal is then to make an inference of the user's encrypted data and send back the encrypted labels to the user. For this task, we used the Optical Recognition of Handwritten Digits dataset from the UCI Machine Learning Library. Since this dataset has an image size of 8x8 pixels, we had to resize the images to 28x28 so as the trained network will accept it as input. Since the images were significantly upscaled from 8x8 to 28x28, the network performed poorly on this dataset. However, this experiment provided a piece of evidence that the data flow of the homomorphic learning works. This experiment also showed that it is rather easy to work with image data for homomorphic learning.

B. Sentiment Analysis

Most of the crucial and beneficial applications of privacy-preserving homomorphic learning are in the area of health care data. We wanted to experiments with one of the simplest of examples in Natural Language Processing. In this experiment, we work with the ISEAR dataset, which a small

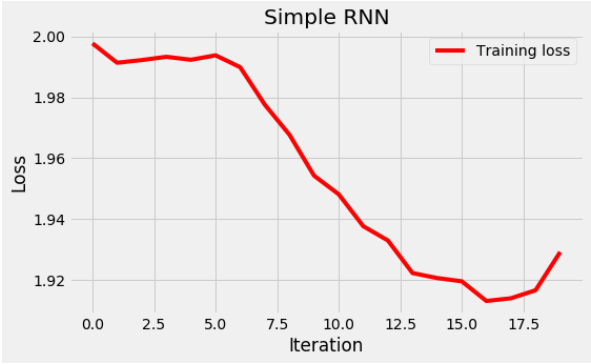

Fig. 3. The performance of Simple RNN on ISEAR dataset

Fig. 4. The performance of Glove embeddings and LSTM on ISEAR dataset

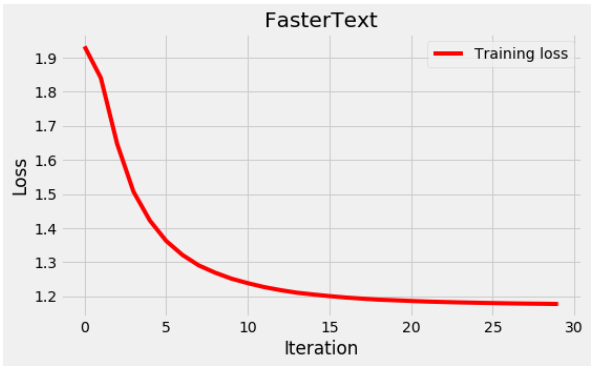

Fig. 5. The performance of 'CNN for NLP' on ISEAR dataset

sentiment dataset with seven unique emotions (joy, fear, anger, sadness, disgust, shame, and guilt) and 7516 different text entries. The dataset comes in a CSV format. The first task is to convert the data in PyTorch torchtext format. We spliced the dataset into training, validation, and testing set with 6764, 375, and 377 texts, respectively. Before training a model with an encrypted *ISEAR* dataset, we wanted to find the best model for this dataset.

We experimented through five candidate models: Simple RNN as a benchmark, LSTM with Glove embeddings, FasterRNN with Bigrams, CNN, and BERT. Based on our experiment, the embeddings obtained from the BERT model combined with GRU and a fully connected layer performed the best with 99.45% accuracy on the training set and 86.05% on the test set.

1) Simple RNN: We used the *torchtext* to define fields for the actual text and the sentiment label. We used *spacy* [18] to tokenize the text. The vocabulary was built with a maximum size of 25000 words. A fairly simple and straightforward RNN architecture was used with an embedding layer that takes the input text, an RNN layer with 256 hidden dimensions, and a final fully connected layer with seven output nodes. This model had 1,036,347 trainable parameters. We used Adam optimizer and CrossEntropyLoss for training the network. This network achieved a maximum accuracy of 18.44% when trained for 20 epochs. Figure 3 shows the performance of the SimpleRNN model on the ISEAR dataset, which is not that impressive.

2) Glove RNN: The vocabulary was built with a maximum size of 25000 words using glove [19] 'glove.6B.100d' vectors trained on six billion words. A fairly simple and straightforward RNN architecture was used with an embedding layer that takes the input text, a bi-directional LSTM layer with 256 hidden dimensions, and a final fully connected layer with seven output nodes. This model had 3,158,435 trainable parameters. We used *Adam* optimizer and *CrossEntropyLoss* for training the network. This network achieved a maximum accuracy of 98.6% on the training set and 80.88% on the test set when trained for 30 epoch. Figure 4 shows the performance of the Glove embedding based RNN model on the ISEAR dataset, which is way better than the base model.

3) Faster RNN: This architecture is based on the 'FastText' [20]. The idea with 'FastText' is to train an RNN model with less trainable parameters and reduce the training time. One of the key concepts of 'FastText' to improve the efficiency of the model is by computing n-grams of an input sentence and append them to the end of a sentence. In our implementation, we will use bi-grams. Briefly, a bi-gram is a pair of words/tokens that appear consecutively within a sentence. The vocabulary was built with a maximum size of 25000 words using 'glove.6B.100d' vectors trained on six billion words. A fairly simple and straightforward architecture with just an embedding layer and a final fully connected layer with seven output nodes is used. This model does not use any RNN units such as LSTM or GRU to make the training faster. This model had 2,500,907 trainable parameters, which is less than the Glove + LSTM model. We used *Adam* optimizer and *CrossEntropyLoss* for training the network. This network achieved a maximum accuracy of 98.6% on the training set and 75.23% on the test set when trained for 30 epoch. Figure 5 shows the performance of 'FastText' on the ISEAR dataset which performance is comparable to the Glove + LSTM model with less trainable parameters and faster training time.

4) CNN: This architecture is based on the 'Convolutional Neural Networks for Sentence Classification' [21]. The idea with 'CNN for NLP' is to use the CNN for NLP tasks rather than images. The intuitive idea behind learning the

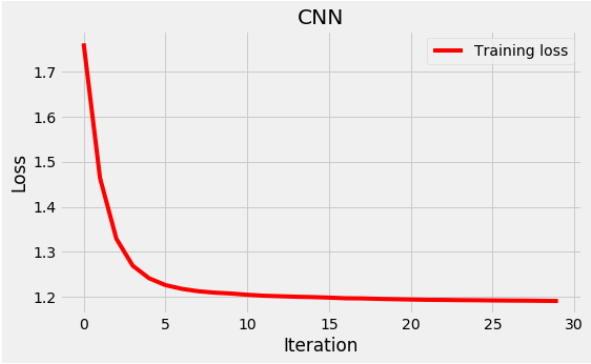

Fig. 6. The performance of 'CNN for NLP' on ISEAR dataset

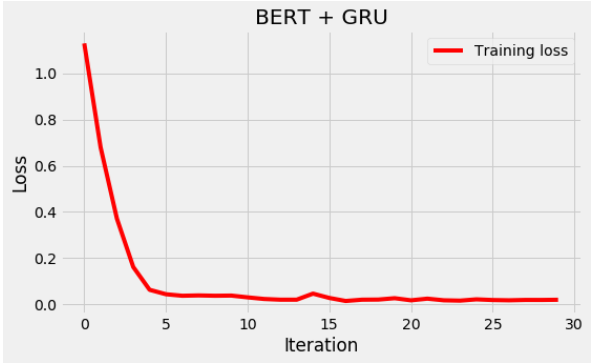

Fig. 7. The performance of BERT + GRU on ISEAR dataset

weights of a CNN layer is that the convolutional layers act like feature extractors, extracting parts of the image that are most important to the network. So CNN's might also extract features from the text? In the same way that a 3x3 filter can look over a patch of an image, a 1x2 filter can look over a 2 sequential words in a piece of text, i.e., a bi-gram. In this CNN model, we use multiple filters of different sizes, which will look at the bi-grams (a 1x2 filter), tri-grams (a 1x3 filter), and/or n-grams (a 1x*n* filter) within the text. The intuition here is that the appearance of certain bi-grams, tri-grams, and n-grams within the text will be a good indication of the final sentiment.

The vocabulary was built with a maximum size of 25000 words using 'glove.6B.100d' vectors trained on six billion words. The 'CNN for NLP' architecture has an embedding layer, three convolution layers, and a final fully connected layer with 7 output nodes. This model does not use any RNN units such as LSTM, or GRU instead uses CNN. This model had 2,592,607 trainable parameters, which is comparable to the Glove + LSTM model. We used *Adam* optimizer and *CrossEntropyLoss* for training the network. This network achieved a maximum accuracy of 97.44% on the training set and 74.99% on the test set when trained for 30 epoch. Figure 6 shows the performance of 'CNN for NLP' on the ISEAR dataset which performance is comparable to the Glove + LSTM model.

5) BERT: This architecture is based on the combination of a transformer model called 'BERT' [22] and bi-direction GRU. Since 'BERT' is trained on a massive amount of publicly available data, it captures intricacies of a language. Instead of using word vectors like 'Glove' or 'word2vec', we utilize the immense power of transformer models. These transformer models are huge and are useless to train on commodity hardware. So we use an already trained 'BERT' model. We used the tokenizer that works with 'BERT' rather than *spacy* tokenizer. Also, the vocabulary provided with the pre-trained *BERT* model is used. Our architecture has *bert* layer that acts as an embedding layer, a GRU layer with 256 hidden dimensions, and a final fully connected layer with 7 output nodes. This model had 2,762,247 trainable parameters and 112,244,487 parameters considering the *bert* model, which bigger than any model we have experimented with. We used *Adam* optimizer and *CrossEntropyLoss* for training the network. This network achieved a maximum accuracy of 99.5% on the training set and 83.18% on test set when trained for 30 epoch, which is the best performance so far. Figure 7 shows the performance of 'CNN for NLP' on the ISEAR dataset which performance is comparable to the Glove + LSTM model.

ACKNOWLEDGMENT

I would like to thank Dr. Wingate for approving this project. Also, I would like to thank my friends from Inspiring Lab for allowing me to use their AWS GPU instances to run the experiments.

REFERENCES

[1] A. Gensler, J. Henze, B. Sick, and N. Raabe, "Deep learning for solar power forecasting—an approach using autoencoder and lstm neural networks," in *2016 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 2016, pp. 002 858–002 865.

[2] E. Tong, A. Zadeh, C. Jones, and L.-P. Morency, "Combating human trafficking with multimodal deep models," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 1547–1556.

[3] K. DelMonico. Anthem pays a record $16 million to settle data breach. [Online]. Available: https://ryortho.com/breaking/anthem-pays-a-record-16-million-to-settle-data-breach/

[4] A. Hern. Fitness tracking app strava gives away location of secret us army bases. [Online]. Available: https://www.theguardian.com/world/2018/jan/28/fitness-tracking-app-gives-away-location-of-secret-us-army-bases

[5] T. V. R. V. D. H. Lente Van Hee, Denny Baert. Google employees are eavesdropping. [Online]. Available: https://www.usatoday.com/story/tech/2019/07/11/google-home-smart-speakers-employees-listen-conversations/1702205001/

[6] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*. Stanford University Stanford, 2009, vol. 20, no. 09.

[7] K. Laine and R. Player, "Simple encrypted arithmetic library-seal (v2. 0)," *Technical report, Technical report*, 2016.

[8] R. A. Hallman, K. Laine, W. Dai, N. Gama, A. J. Malozemoff, Y. Polyakov, and S. Carpov, "Building applications with homomorphic encryption," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 2160–2162.

[9] S. Halevi and V. Shoup, "Algorithms in helib," in *Annual Cryptology Conference*. Springer, 2014, pp. 554–571.

[10] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017*, 2018.

[11] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "ngraph-he2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. ACM, 2019, pp. 45–56.

[12] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, 2016, pp. 201–210.

[13] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[14] A. A. Badawi, J. Chao, J. Lin, C. F. Mun, S. J. Jie, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, "The alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus," *arXiv preprint arXiv:1811.00778*, 2018.

[15] Q. Lou, B. Feng, G. C. Fox, and L. Jiang, "Glyph: Fast and accurately training deep neural networks on encrypted data," *arXiv preprint arXiv:1911.07101*, 2019.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] L. v. d. M. A. B. Mark Ibrahim, Shubho Sengupta, "A framework for privacy preserving machine learning," https://github.com/facebookresearch/CrypTen, 2019.

[18] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.

[19] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[20] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[21] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.