Assignment 4

Question 1.

For this question, I used the library Tensorflow encrypted to implement a two party privacy preserving logistic regression. As discussed in the office hours on Thursday(04/22/2021), I have changed the sigmoid function to the piecewise function to make the logistic regression work in the way the question describes it. The link below links to the library -

https://github.com/tf-encrypted/tf-encrypted

TF-encrypted has an implementation of privacy preserving logistic regression. Using the same, I assumed two parties, Alice and Bob that want to implement the logistic regression.
The dataset contains 7000 samples with 32 features, 16 are held by Alice and 16 are with Bob. Both of the datasets are included in the folder. The goal is to detect fraud using this data(Let's say Alice is a bank and Bob is a government and both have data from common individuals).
In the folder I have enclosed we have -

training_alice.py
Implements the Logistic Regression from Alice's side.

training_bob.py
Implements the Logistic Regression from Bob's side.

training_server.py
Implements the server side logic.

common.py
This file implements the logistic regression.

pond.py
This is a standard file in the library which can be found at
https://github.com/tf-encrypted/tf-encrypted/blob/master/tf_encrypted/protocol/pond/pond.py#L1215
I have modified this file and added the function apply_piecewise(x) which accepts a Tensor and applies the piecewise function to each element of the tensor. This function replaces the sigmoid() function in the original code.

Note: This file is directly edited from the library folder, the copy attached here is just to show the code. To run it, this file must be placed in the library path.

aliceTrainFile.csv
Training data from Alice.

bobTrainFileWithLabel.csv
Training data from Bob, along with the class label for fraud.

config.json
Config contains the ip addresses of Alice, Bob and the Server. I ran this on my local machine hence I have assigned localhost with different ports for each of these.

Results-
This is the implementation of the piecewise function. The function takes in a PonsPrivateTensor, converts it into a Numpy array, applies the piecewise function to each element, converts it back into a PondPrivateTensor(This type of tensor is required by the library) and returns the same.

```
def apply_piecewise(x):
    import tf_encrypted as tfe
    import numpy as np
    import tensorflow as tf
    from tf_encrypted.keras import activations
    #from tf_encrypted.keras.activations import sigmoid
    #from util import sigmoid
    import sys
    sys.path.append('/Users/abhineethmishra/Documents/CSE_598_SML/assignment_4/tfe/tf_encrypted/protocol/pond/')
    #from pond import sigmoid

    #Convert tf tensor to PondPrivateTensor
    def createPondPrivateTensor(tf_tensor):
        @tfe.local_computation
        def provide_input():
            #return tf.convert_to_tensor([[1,0.4,-0.7,0.2],[0,0,1,1]],dtype=float)
            return tf_tensor#tf.ones(shape=(5, 10))

        # define inputs
        x = provide_input(player_name='input-provider-0')
        return x
    #w = provide_input(player_name='input-provider-1')

    #Test variable -- substituted as the function argument
    #x = createPondPrivateTensor(tf.ones(shape=(5, 10)))

    #returns a numpy array from a PondPrivateTensor
    def PPT_to_numpy(tensor):
        return tf.keras.backend.get_value(x.reveal().to_native())


    #Implementing the function in the question
    def piecewise(number):
        #print(number)
        number = float(number)
        if number < -0.5:
```

```
    #Test variable -- substituted as the function argument
    #x = createPondPrivateTensor(tf.ones(shape=(5, 10)))

    #returns a numpy array from a PondPrivateTensor
    def PPT_to_numpy(tensor):
        return tf.keras.backend.get_value(x.reveal().to_native())


    #Implementing the function in the question
    def piecewise(number):
        #print(number)
        number = float(number)
        if number < -0.5:
            #print(number,'return 0')
            return 0
        elif number >= -0.5 and number < 0.5:
            #print(number,'return ',0.5+float(number))
            return (0.5+float(number))
        elif number>=0.5:
            #print(number,'return 1')
            return 1
    func = np.vectorize(piecewise)
    #print(PPT_to_numpy(x))
    a = func(PPT_to_numpy(x))
    #print(a)
    np_array = PPT_to_numpy(x)
    print(np_array)
    for index, value in np.ndenumerate(np_array):
        np_array[index[0],index[1]] = piecewise(value)
    applied_function_tensor = tf.convert_to_tensor(np_array)
    print(np_array,createPondPrivateTensor(applied_function_tensor))
    return createPondPrivateTensor(applied_function_tensor)
      # operate here
```

```
(tfe) MacBook-Air:assignment_4 abhineethmishra$ python training_alice.py
Falling back to insecure randomness since the required custom op could not be found for the installed v
ersion of TensorFlow. Fix this by compiling custom ops. Missing file was '/Users/abhineethmishra/minico
nda3/envs/tfe/lib/python3.6/site-packages/tf_encrypted/operations/secure_random/secure_random_module_tf
_1.12.0.so'
WARNING:root:Falling back to using int100 tensors due to lack of int64 support. Performance may be impr
oved by installing a version of TensorFlow supporting this (1.13+ or custom build).
Starting alice...
WARNING:root:Falling back to using int100 tensors due to lack of int64 support. Performance may be impr
oved by installing a version of TensorFlow supporting this (1.13+ or custom build).
2021-04-23 21:18:02.753312: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports inst
ructions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2
2021-04-23 21:18:02.755254: I tensorflow/core/distributed_runtime/rpc/grpc_channel.cc:222] Initialize G
rpcChannelCache for job tfe -> {0 -> localhost:5011, 1 -> localhost:5018, 2 -> localhost:5019}
2021-04-23 21:18:02.756091: I tensorflow/core/distributed_runtime/rpc/grpc_server_lib.cc:381] Started s
erver with target: grpc://localhost:5011
WARNING:root:Falling back to using int100 tensors due to lack of int64 support. Performance may be impr
oved by installing a version of TensorFlow supporting this (1.13+ or custom build).
2021-04-23 21:18:04.458578: I tensorflow/core/distributed_runtime/master_session.cc:1161] Start master
session 44739402304047e3 with config: graph_options { }
> /Users/abhineethmishra/Documents/CSE_598_SML/assignment_4/common.py(26)forward()
-> y = tfe.apply_piecewise(out)
(Pdb) c
[[-0.03540039]
 [-0.04421997]
 [-0.06080627]
 [-0.08068848]
 [-0.01408386]
 [-0.06872559]
```

```
Batch  329
Batch  330
Batch  331
Batch  332
Batch  333
Batch  334
Batch  335
Batch  336
Batch  337
Batch  338
Batch  339
Batch  340
Batch  341
Batch  342
Batch  343
Batch  344
Batch  345
Batch  346
Batch  347
Batch  348
Batch  349
Weights on alice: ((([0.1764373779296875]
 [0.2728729248046875]
 [-0.0032196044921875]
 ...
 [0.2723846435546875]
 [0.0037994384765625]
 [-0.013092041015625]], [0.264129638671875]),)
(tfe) MacBook-Air:assignment_4 abhineethmishra$
```

To run this, all three files training_alice.py, training_bob.py and training_server.py have to be run in three terminal instances. The model is trained for 350 batches and the trained logistic regression model (weights for each feature) is printed on the terminal (The screenshot shows the output at Alice's terminal).

Question 3.
POSEIDON: Privacy Preserving Federated Neural Network Learning

There are two traditional ways to train a model, centralized and decentralized MPC, where the former has no privacy and the latter needs an honest majority assumption plus only 4 parties can train the model. In Federated Learning, many parties can participate and they don't have to share their data. Each party agrees to a NN architecture and trains the model locally, and then shares the gradients with the server. The server will calculate the mean to the gradients and get a global model. Now the paper uses oblivious inference, where it does key switching to make sure the servers don't extract the query output and the output is only visible to the querying party. This paper also introduces alternative packing which is used to decrease the number of rotations, since rotation is a costly operation(The paper improves complexity from linear to

logarithmic). The paper also proposes a new method for bootstrapping, which is more efficient than traditional centralized bootstrapping.
This paper uses homomorphic encryption, which was discussed in class

Some new techniques discussed in the presentation are -

Differential Privacy - DP adds mathematical noise to a small sample of the dataset to improve privacy. If a high privacy guarantee is used, the model accuracy will be affected.

Bootstrapping - Used to solve the multiplication problem (Being able to perform homomorphic multiplication without the layer limitation).

Approximated Activation Functions - Since ciphertexts are encrypted, the authors show ways to approximate Sigmoid and ReLU activation functions.

POSEIDON solves the model confidentiality problem so the other parties cannot learn the model weights.

Results: POSEIDON achieves accuracy close to centralized training. With respect to the performance, it scales linearly with the number of features, due to the increased communication cost. Scales logarithmically with the number of layers.