3.2 Dropout

Dropout networks were first proposed by Srivastava et. al to regularize training neural networks. Each iteration during training, a predefined percentage of neurons in the network are randomly disabled and receive zero gradients. The paper experiments with keeping input layer neurons 80\% of the time and dropping hidden layer neurons 50\% of the time. The idea behind disabling neurons during training is to force individual neurons to learn meaningful representations instead of co-adapting (relying on other neurons in the network for meaningful information). <Mention ensemble averaging> They found by training networks with dropout and then scaling the hidden connections during inference by the number of neurons in the layer <check this>, the could improve generalization by <p\%>.

We mention dropout networks because they pose as a good starting point for developing the intuition for why the Shokri technique works and for how we considered handling dynamic computation graphs.

SMPL is based on the Shokri technique. SMPL takes that to a peer-to-peer setting using mpc. In their technique and many other distributed training techniques, they assume that the topology of network is known to all party members and that the topology is static. To further generalize our technique, we want to be able to handle dynamic computation graphs so we can accommodate arbitrarily-shaped data. Just as in the Shokri paper, each iteration, we choose a percentage of the gradients to share with the gradient server. If we naively share gradients without taking to care to secret share the shape of the data, we will leak information about each party’s dataset. Take for example that there three hospitals that would like to train on patient health records to train their own neural networks to learn cancer cell detection and would like to use secure multi-party learning to share data to improve everyone’s networks. The datasets are all assumed to come from the same domain and share a consistent format, however, some records may contain additional information that are intrinsic to particular types of patients or even individual patients. This data can still be used in the training sets as long as those unique features have their own index in the input vectors and that’s agreed upon by all party members. As an example, they all agree that the first index in the input vector represents patient height, the second represents patient weight, etc. The same applies to any other domains such as image, audio, and text data, etc. On each iteration, as batches are fed through the network, the number of gradients shared changes since we are taking a percentage of the number of parameters in the network at that iteration. The number of gradients shared can increase or decrease across iterations. When gradients are shared, the requester asks for particular coordinates. An adversary can record the coordinates requested over the entire training phase and use the frequency of requests per coordinate to determine the upper bound on the shape of the network overall, but can also directly observe the shape of the input layer and output layer as they change over time and determine the shape of the largest input. Although revealing the shape of the input and output may seem innocuous, this information could be used to infer what type of data is in a particular member’s dataset. Going back to the hospital example, imagine an adversary knew that one of the hospitals had a particular patient with a particular pre-existing condition and would like to use this to discriminate against that patient. The adversary could join the party and observe the requested coordinates as mentioned above, plot the distribution of requests for each coordinate from each member in the party and discern the upper bound on the input layer and output layer of their networks. Coordinates are never secrete shared so that party members can align their networks so that they can respond with the appropriate gradients - graph alignment problem. Thus, the adversary knows that for a feedforward network, coordinate [l, r, c] corresponds to layer l, row r, column c e.g. [0, 0, 0] is refers to the gradient in the first hidden layer, in the first row, first column of the matrix. If [0, 0, 100] is the largest coordinate from the first layer, then they know that the requester’s dataset contains a vector that contains at most 100 features, and because the format of the vectors (format of the datasets) must be agreed upon, the adversary then knows that particular hospital’s dataset contains a patient with those attributes (height, weight, …, has cancer or not).

A possible remedy that we considered was to embed the network in a faux parameter space. This idea was inspired by dropout networks and how each iteration the dimensions of the network changes and the static topology acts as the upper bound on the shape of the network. The idea was to choose a large random N, the upper bound of the faux parameter space, and a fake percentage f, different from the p percent - true percent - of gradients to share, such that N\*f is greater than the greatest number of parameters that could be possibly shared throughout the training phase. Each iteration, a different N and f could be chosen to create a noisy upper bound. The lower bound of the shape of the network is known to all party members - all networks must have at least one input, and one output neuron, or effectively a lower bound of 0 parameters, so that doesn’t need to be hidden. After choosing the faux upper bound, coordinates could be chosen uniformly at random across the faux parameter space to share mixing both true coordinates and faux coordinates. Faux gradients would also be generated from the same distribution as the noise used to mask the true gradients.

The first problem and most obvious problem is that if N is too large, the parameter space could be unrealistic and would effectively hide nothing because everyone including the adversary knows the domain of the data and the architecture agreed upon by all members. Additionally, currently the largest network ever trained is 1000 layers deep with <n-parameters> and generating and sending these faux coordinates and points would incur a large overhead.

For this approach to work, the faux parameter space would need to be large enough to seem real to an adversary so that would not be able to distinguish between real and faux data. This would mean that the noisy upper bound would need to be large enough to hide the largest possible set of gradients to be shared while not being unrealistically large.

However, even if we could mask real and faux points, that does not address targeting the coordinates in the input layer. In this type of attack, the adversary would only need to guess along one dimension of the parameter space to infer the shape of the largest datapoint in a dataset. In a practical setting, this would be trivial since the number of differing attributes is finite and most likely small. For example, if the networks are training on images, realistically we know that most data points will be 400x300, 640x480, 800x600, 1024x768, etc. and so the difference between input vectors are bounded by dimensions, bounded by computation resources, and standardized data formats. The difference between patient health records are most likely even more bounded than images.

The second problem with the faux parameter space approach is frequency of updates to coordinates. Certain gradients would be updated more often than others. <explain more about these papers> In order to hide the shape, we should need to sample random point uniformly across the parameters space to hide which these parameters. As the number of parameters in the network grows, the size of the sampled space would also grow meaning we would need to sample and send $O(n^2)$ points each iteration so that the density of points in a particular location appears not different from any other location in the parameter space. This would not scale given the network bottle. If the adversary were able to distinguish between real and fake points, they could start to reconstruct the shape of a party member’s network and infer the type of points in their dataset. One avenue we considered to address that would be to send and respond to gradients requests with a set of points with a uniform number of fake and real points and occasionally only send fake batches. However, neither of these address targeting the inputs layer dimensions.

We also considered cryptographic techniques like ORAM and possibly other oblivious data structures, but concluded that the computational demand would impede the practicality of training under secure multi-party learning. We leave this to future research.