3 Preliminary Ideas

This section presents preliminary ideas we had and conclusions we drew about protecting each party’s computation graph and for improving distributed training performance.

3.1 Dynamic Computation Graphs

At a primitive level, neural networks can be view as computation graphs - a graph representing the composition of operations performed by a learning algorithm. To handle variable-sized data and situations, neural networks can change shapes, which are commonly referred to as dynamic computation graphs (DCG) e.g. tree-lstms, dropout networks, dropconnect networks, freezeout networks, dense-sparse-dense networks, networks evolved through augmenting topologies [cite all neat papers]. DCGs require optimized batching algorithms to group similarly shaped input together during training. Gradients are backpropagated in the same manner as static computation graphs.

One of the ideas we initially considered to generalize our approach was to protect the dimensions of the computation graph. During distributed training, the model parameters must be synchronized with all parties. When training with dynamically-shaped data, a new computation graph is created different input dimensions. The gradients are then batched together to update a single model. Synchronizing DCGs with the other party members reveals information specific to that particular member’s dataset. For instance, in collaborative training, everyone must agree on the same input data domain and perhaps these are medical records and perhaps these medical records can be variable length, but the average patient’s record is 10-dimensional. However, some patients with particular diseases have 11-dimensional records indicating their disease. Furthermore, assume that only one of the hospitals has records on a handful of patients with patients with this disease, but most of the records this hospital has appear typical. All hospitals agree on training with DCGs. During local training, that particular hospital samples a batch to train on and it just so happens that there’s an 11-dimensional vector in that batch, so a DCG is created with an eleven-dimensional input layer, which corresponds to the first weight matrix having eleven columns. When the parameters are synchronized, the other parties can now observe that this hospital’s input layer contains eleven dimensions indicating that their dataset contains some patients with that disease.

Perhaps to address this problem, we could just share individual coordinates. However, this still requires revealing that coordinates in the eleventh column. An idea we explored was to obfuscate the coordinates by pretending that our parameters are actually in a higher dimensional space and then sampling from that imaginary coordinate space uniformly because certain activations are more likely than others. Intuitively, our matrices would be padded around the edges with some margin of space to hide our true input dimensions. Although this was an interesting, idea, there are two main problems. First, to collaboratively train, coordinates must align across all parties, so for example, the first parameter $\theta\_{00}^{a}$ in the first layer’s weight matrix can only replace the value in the first layer’s weight matrix $\theta\_{00}^{b}$. Thus, at some point the coordinates must be compared to know that they are replacing the correct positions. Incorrectly synchronizing parameters will lead to the network forgetting particular features. Second, all parties know that only particular coordinates can be real because all parties already agreed on the input domain. If the input are again medical records, the padding cannot be millions of features when all parties already know that the average medical record, in this case, only contains 10 features. Adding this extra margin means maintaining unnecessary state, additional communication, and computation.

//Add picture of idea

In the next section, we give some background on a type of dynamic computation graph used for regularizing training.