1. Good morning everyone. Today, I'll be presenting my BTP work on Federated Learning that I had performed under the supervision of Prof. Bharath during this semester. The title of the presentation is: "Omni-Fedge", which is a federated algorithm we have proposed following a Bayesian approach. Let us start.

2. What is Federated Learning? It is a distributed machine learning architecture where edge-devices learn shared predictive model collaboratively. As we see here, an edge-device downloads the current model, improves it by learning from its local data, and then summarizes the changes as a small focused update. Only this update to the model is sent to the federating server in the cloud, where it is processed with updates from other edge-devices to improve the shared model. All the data remains on the edge-device, hence privacy is preserved.

3. Google currently uses FL in their Android GBoard to predict the next query a user makes based on the words he types. Federated Learning actually started as a project at Google in 2017.

4. In many European Union countries, patient privacy laws are stringent. It is legally burdensome to send patient data such as MRI-scans, X-ray films or blood-samples to the cloud for diagnostic services based on machine learning algorithms. In such scenarios, federated learning can be employed for private learning among various hospitals, with the data staying with the hospitals itself.

5. Another promising application of federated learning is self-driving vehicles in an autonomous vehicle network. Let us say that in near future we have self-driving cars on roads, and the cluster of these cars within a geographical perimeter can collaboratively learn the traffic conditions.

6. Since we have now seen some potential applications of federated learning, let us now see what challenges lie ahead of us before we bring it into implementation.

7. First challenge is about stragglers in the edge-devices. Let us take the example of smartphones learning a task collaboratively. Some phones may not be available at the time the learning is happening. In such cases, how does the model that is learnt perform due to the unavailability of these phones?

8. Next is the assumption that the data available at edge-devices in non-IID. The assumption that data is IID makes analysis simpler, however, doing so may incur heavy errors in learning. In principle, that data available at different edge-devices may follow different distributions altogether.

9. We then have device heterogeneity. As an example, smartphones vary in their memory, processor capacity and network bandwidth available to them. They are all not the same. How do we account for this while creating our model?

10. Since federated learning is a distributed machine learning architecture, there will be lots of communication taking place between various entities in the network. So communication costs will be high. How do we optimise the communication?

11. Finally, we have the privacy concerns. Though federated learning is privacy preserving by default, how do we ensure that the updates cannot be re-traced to identify the users? How do we encrypt them?

12. Having understood what lies ahead of us, let me now begin with the technical part of the presentation.

13. Let us recall what a neural network is. We have training data available with us, labelled as z11, z12,…,z1n.

14. The neural network assigns weights to the data, which can be thought of as some knobs which are to be tuned.

15. 16. Then the objective is to minimise a function of the data and the weights, such that the resulting penalty (or loss) is minimum.

17. Let us say that we have determined the weights which minimise the loss, analogous to finding the right positions of all the knobs.

18. 19. 20. Then we test the neural network by feeding in unseen data and check how good our model performs the task.

21. Having recalled what the neural network is, let us now understand how federated learning is formulated. We take the example of 3 smartphones all throughout the presentation henceforth. Each phone has local data denoted by z.

22. Also, each phone has a local objective f that it wishes to minimise. An example of f can be a 0 - 1 loss in query prediction. If the user clicks on the predicted query, assign 0 to it and 1 otherwise.

23. Each phone then does a local training to find the best neural network weights which minimise the loss.

24. Let us denote those by \theta^{*}

25. These weights are sent across the network to the federating server.

26. The federating server then processes these weights pooled from all the phones to create a new update.

27. This update is broadcasted back to the phones.

28. The phones then plug-in this update and learn the shared model.

29. An example shown here is of next word prediction. Note here that if we take g as the average of weights, what we have seen just now turns out to be the big picture of a well known FL algorithm called Federated Averaging, or FedAverage in short.

30. Let me now explain the conditions that we have considered in our problem. We have assumed the local data to be non-IID.

31. Further, we assume that the local data is sampled from a distribution D.

32. Each device wishes to minimise the expected value of the true loss with respect to the neural network weights.

33. However, the true loss is unknown as the distribution of the data itself is unknown!

34. So, we instead use a proxy for the true loss. We do so by replacing the true loss with an empirical estimate of the true loss, which we would be referring to as the empirical loss. This empirical loss can be squared error loss, cross-entropy loss or any other estimate. The losses here are treated as random variables since they are a function of the data points sampled randomly from a distribution.

35. We now observe that if the neural network is same across all the devices, we will be losing out on crucial local information altogether, which is not so desirable.

36. Hence, we split the neural network into two: shared and task-specific layers, denoted by \theta^{(1)} and \theta^{(sh)} respectively.

37. The shared layers, as the name suggests, are common across all the edge-devices. The shared neural network weights will be learnt in a manner that minimises the overall error while the task-specific ones will be fine tuned to meet edge-device specific requirements.

38. I'll now talk about the Bayesian approach that we have taken. Let us say that the phone with a tick-mark is the perspective through which we talk.

39. We assume that the loss at our phone follows an exponential distribution with \omega as the parameter. Here we interpret \omega as the weight that our phone gives to itself and others based on the statistical similarity of the local data available to it and the other phones. The intuition is as follows: if the data is IID, then the weights assigned to each phone will be a simple average; however, if the data is non-IID, which is our case, then we assign more weight to that phone whose local data is statistically closer to the local data available at our phone. Here, we decide upon the closeness of the local data at another phone by testing how well the model on our phone works on other phone. Since the shared layers are common across all the phones, this boils down to whether the neural network resulted by using task-specific layers of our phone works well on the other phone or not, which is gauged by the loss.

40. The joint distribution of losses as observed by our phone is shown here, which we represented by Q. The true joint distribution is represented by P.

41. - 46. Repetition through different perspectives.

47. So, our objective now boils down to finding a suitable method to assign weights to different phones based on this intuition of ours. This results in the following problem. For a given \theta^{(i)} and \theta^{(sh)}, we minimise the following objective function with respect to the weights \omega_{l}. The first term of the objective is a weighted average of the empirical losses and the second term is a regulariser which prevents \omega_{l} from taking extreme values. Our Omni-Fedge algorithm roughly does the following:
First, we start with an initial value of \theta^{(sh)}, and minimise the empirical loss with respect to \theta^{(i)} based on the local data alone. We get the best value of \theta^{(i)} in doing so. This step corresponds to fine-tuning the neural network to the local-data. We use this \theta^{(i)} along with \theta^{(sh)} to find the optimal weights to be assigned to the phones, by minimising the objective shown here. This step captures the statistical similarity concept that I mentioned just a while ago. We next use the optimal weights

along with the best \theta^{(i)} to find that \theta^{(sh)} value which minimises the error across all the phones. This step realises the shared learning part. All these three steps are repeated until we get a good accuracy. I will explain the working of the algorithm through an example shortly. But before I do that, I need to gauge the performance of the neural network obtained by solving the above problem in comparison with the true optimisation problem, for which I will need to introduce a complexity measure.

48. R_{I}(\theta) is what we call the log-exp complexity of the neural network. In the numerator we have the exponential of the expected value of the true loss and in the denominator we have the product of the empirical losses. We take the supremum of this ratio over the neural networks weights, find its expected value over the distribution Q and take its logarithm. This measure though which at first seems magical, crops out of the mathematics when we apply McDiarmid's theorem to find the PAC bound. As of now, for the sake of analysis, I could tell that higher this term, more is the error incurred by our algorithm.

49. I now present the main theorem of our work which measures how much error our algorithm incurs. It suggests that the true average loss is no more than our proposed objective and the error term. The error term is in terms of KL divergence between the exponential distribution Q and the true joint distribution P. It says that if the true distribution is close to the independent exponential distribution, then the error in terms of KL divergence is small, as expected. Further, the error also depends on the log–exp complexity. The last term in this equation indicates that higher weights correspond to higher error. So we are now in a comfortable position to state the algorithm and understand how it works, as we are assured that it works indeed!

50. Here is our Omni-Fedge algorithm. As discussed sometime earlier, the algorithm has three major steps, part of which needs to be carried out in a distributed fashion, hence it looks complex. But to give an idea, I will now show the working of the algorithm taking a small example with three phones, as I have been doing so far.

51. First, each phone computes the best value of \theta^{(t)} for which the empirical loss is minimised based on the local data alone.

52. To 56. These values are broadcasted by each phone to the other two phones through the federating server. Please note the colour-scheme that I have followed here for a more lucid understanding.

57. Next, each phone computes its empirical loss based on its local data using the task-specific neural network weights of all the three phones, including itself.

58. To 61. Then each phone sends the corresponding empirical loss value to the respective phones. At this stage, each phone has the empirical loss value obtained by using its neural network weights on itself and on the other phones as well. Since each phone now has the information on how well its neural network has performed on its data and on the other phones, it could judge how statistically closer are the other two phones to it and assign the weights accordingly.

62. It does so by minimising the objective function that we had discussed earlier.

63. - 67. Now each phone broadcasts the weights it had computed to the other two phones through the federating server. At the end of this communication, each phone has all the weights that were computed by itself and the other two phones.

68. Now comes the shared learning step. Each phone now computes the gradient of the empirical loss based on its local data by using the neural network weights of itself and the other two phones. It then weighs them accordingly using the weights obtained in the earlier step and sums them up. Thus, each phone computes a part of the gradient of the overall empirical loss.

69. - 73. Each phone then broadcasts these partial gradients to the other two phones through the federating server. At the end of this communication, each phone has all the partial gradients that were computed by itself and the other two phones.

74. Each phone then updates the shared neural network weights accordingly as it now has all the information required for the update.

75. All these steps are repeated until either convergence or a good average accuracy is reached on all the phones. In this manner, the algorithm takes the best of both the worlds of task-specific and shared layers of the neural network. Practically, if we were to take the query prediction example, this would mean that the prediction knows the nuances of your style of writing, yet it could correctly predict the global trend about which you are possibly querying.

76. I will now quickly share our experimental setup and results. We have implemented the nodes and server as callable objects in Python, and used Tensorflow API for implementing the neural network. Here we do not explicitly communicate through sockets, but emulate the same using the nodes and server objects, and a driver code which orchestrates the various steps of our algorithm.
The experiments were carried out using MNIST handwritten data set and FMNIST fashion data set, with 5 nodes and 1 server. Both MNIST and FMNIST have 60,000 training examples and a test set of 10,000 examples each. Each example is a 28×28 grayscale image, associated with labels from 10 classes. The table here shows the split between training and test data. The non-IID case was emulated using a non-uniform sampling of data from the MNIST and FMNIST data sets. For the IID case, we ensured that roughly equal number of samples from each class were included. The data samples assigned to each node were further divided into 100 batches, and batch-wise training was performed. Our algorithm was then compared with local training, i.e., training using local data only, and FedSGD, which is a slight variation of FedAvg. While implementing FedSGD, the gradient was averaged, and one neural network was used across all the devices. Spare-categorical cross-entropy loss was used all throughout in these experiments.

77. All the plots show average of the accuracies across 5 nodes on the y-axis for Omni-Fedge, FedSGD and local training. On the x-axis we have the number of communication rounds, where each communication round constitutes the three steps of Omni-Fedge as discussed earlier. Recall that these three steps amount to one update of all the neural network weights. This plot is for non-IID data using MNIST.

78. This plot is for IID data using MNIST. We see here that the algorithm starts with a good accuracy initially.

79. This plot is for non-IID data using FMNIST. We see a similar trend to that of non-IID data using MNIST, except that the variation is higher here.

80. This plot is for IID data using FMNIST. It is again similar to that of IID data using MNIST, except that the variation is higher here too.

81. This plot shows the running average for IID data using MNIST and FMNIST, doing away with the variation. Here we see that Omni-Fedge achieves greater accuracy on MNIST when compared to FMNIST for same number of communication rounds, and is understandable due to the more complex nature of FMNIST images than MNIST.

82. To summarise the results, Omni-Fedge clearly outperforms local training and FedSGD for both IID and non-IID data. We have also observed that it assigns higher weights for more relevant nodes under non-IID case, and assigns equal weights to all the nodes under IID case, the value being 0.2 since we have taken 5 nodes.

83. To summarise the work, we considered a specific FL problem under a non-IID data setting. We then provided a sound theoretical framework for the proposed algorithm based on a novel Bayesian approach. We also introduced a new complexity measure as a consequence of the PAC bound that we had obtained.

84. We submitted a paper titled “Federated Algorithm With Bayesian Approach: Omni-Fedge” to ICASSP 2021, whose result is awaited. Prof. Bharath and I are also working on a journal paper in this regard. We have also analysed about the stragglers, and found out that they are addressed in this framework that we have laid. Also, the log-exp complexity is almost understood when we analyse the nature of the function and the constraints that we impose. What we are yet to do is analyse the convergence of Omni-Fedge and think about how we can make it more communication efficient, like using signed-gradients, quantised gradients, and reducing communication to a greater extent if possible.

85. - .87 Here are the references.

88. Thank you. It is time for questions.

I would like to thank Prof. Bharath for providing me this opportunity to work with him in his team and I am glad to inform you all that I will be continuing this BTP in the next semester also.