UNIVERSITY OF CALIFORNIA SAN DIEGO

Bayesian Decentralized Learning

A Thesis submitted in partial satisfaction of the requirements
for the degree of Master of Science

in

Electrical Engineering (Machine Learning and Data Science)

by

Osman Cihan Kilinc

2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Farinaz Koushanfar for her support as the chair of my committee, supervision, teachings, encouragement and remarks throughout the writing of this thesis. Besides my advisor, I would also like to thank Anusha Lalitha and Professor Tara Javidi for their invaluable contributions to the research explained in this work. Even though we may have found each other on complete opposite ends of the spectrum, it was their support that steered me in the right direction and introduced me to many new topics in the field.

I want to thank all my friends at the ACES lab for making me look forward to arriving to the lab in the morning: Mohammad Samragh Razlighi, Mojan Javaheripi, Huili Chen, Siam Umar Hussein, Malhar Jere, Xavier Carpent and Bita Darvish Rouhani. Outside the lab, I want to thank all the friends that I have made throughout Master's for their camaraderie. They truly made the last two years unforgettably wonderful: Kazim Ergun, Apsara Williams, Austin Parsons, Rhea-Comfort Addo, Olcay Soyalan, Ray Berkeley, Can Uysalel, Gokce Sarar, Aditya Vasan, Kyle Thackston, Corentin Pochet, Viona Deconinck, Vanessa Kelly. I want to thank the Bayraktar family, Canan Bayraktar, Ozdemir Bayraktar, Selcuk Bayraktar, Haluk Bayraktar and Ahmet Bayraktar for showing me what's possible with hard-work and being great role models.

Finally and most importantly, I would like to express my deepest gratitude to my parents. Nothing would be possible without them and their support. Thank you.

ABSTRACT OF THE THESIS

Bayesian Decentralized Learning

by

Osman Cihan Kilinc

Master of Science in Electrical Engineering (Machine Learning and Data Science)

University of California San Diego, 2019

Professor Farinaz Koushanfar, Chair

The field of machine learning has grown tremendously in the past decade. It is utilized in many different industries with applications ranging from high-tech security systems to medical diagnosis. In order to train their models, technology companies aggregate vast amounts of data from their users and execute training in big data centers. In such an era, where the means of machine intelligence is gathered at the hands of the few and data privacy rights are continuously breached, the need for a more democratic and privacy-preserving machine learning method is exigent. Fortunately, advances in mobile computing is gradually moving the computations on the cloud to the devices. Decentralized learning reinforces these advances by enabling training on decentralized data. In decentralized learning, users train their models with their local datasets and share the acquired knowledge with each other. It mitigates data-sharing and provides a degree of freedom for model personalization. Thus, research on decentralized learning has gained pace. In this thesis, we explore decentralized learning, make an analysis from a bayesian perspective, explain the relevance of continual learning, and demonstrate empirical results from an up-and-coming decentralized learning method.

Chapter 1

Introduction

Humans have an innate ability to learn new knowledge. However, our individual intellectual capabilities are limited to a large degree and most of our knowledge comes from the wisdom gained over the centuries of human progress. Every generation inherits the accumulated knowledge from the previous generations. This knowledge is vital for our survival and it constructs the modern civilization. In other words, the quintessence of our civilization lies in our individual and collective ability to learn, accumulate and transfer knowledge.

The collective intelligence emerges from our collective efforts to solve problems [1]. Open source projects are wonderful examples to such collaborations[2]. They are continuously improved and tested. Thus, knowingly or not, billions of people benefit from these projects. Similarly, a collective machine intelligence can provide an efficient, accurate and robust solution to a myriad of problems [3]. However, we are yet to discover a truly decentralized and effective method to accumulate machine intelligence. The ultimate goal of this thesis is to provide a detailed analysis of the decentralized learning problem and demonstrate the capabilities of a decentralized learning method that utilizes variational inference. Before moving on to decentralized learning, it is vital to briefly visit the basics of machine learning.

Machine Learning

By definition, machine learning is the scientific study of creating, training and analyzing statistical models that has the ability to acquire new knowledge[4]. These statistical models mathematically describe phenomena and their related uncertainty in terms of probability distributions over latent variables or unknown parameters. Essentially, they are used to make an inference, where we intend to minimize the expected risk involved in decision making. The risk function uses a loss function to compute the error between a prediction and the desired result, denoted by $L(y,\hat{y})$, where y is the prediction and $\hat{y}$ is the true value. Using y to denote a label and x to denote the input vector, let's describe a classification problem as an example. Suppose $g(x)$ is a decision function that makes the prediction $y \in Y$ given $x \in X$. Then, we can write the risk and the output of our model ($\hat{g}(x)$) as:

$$\hat{g}(x) = \underset{g(x)\in Y}{\arg\min} E_{x,y}[L(g(x),y)] \tag{1.1}$$

$$E_{x,y}[L(g(x),y)] = \int \sum_{y\in Y} p(x,y)L(g(x),y)dx \tag{1.2}$$

Risk1 can be also written as:

$$E_{x,y}[L(g(x),y)] = \int p(x) \sum_{y\in Y} p(y|x)L(g(x),y)dx \tag{1.3}$$

Under "0-1" loss, the class with the highest probability is chosen, which is:

$$\hat{g}(x) = \underset{y\in Y}{\arg\max}\, p(y|x) \tag{1.4}$$

Thus, probability of y given x should be closely approximated to make an accurate prediction. This is achieved through training. In practice, data represents the phenomena so we wish to encode the information on the distribution of data into the statistical models such as $p(y|x)$. In other words, we are trying to learn the parameters or latent variables that best explains the observed data so that we can use them for unobserved data that comes from the same underlying distribution. This leads us to the question of parameter estimation. How do we incorporate all available information? How can we encode prior knowledge? In the context of decentralized learning, how can we convey what's learnt by an individual to all other participants?

In Bayesian learning [5], training is built upon a prior distribution, $p(w)$, which is transformed into a posterior distribution (after observing data), $p(w|D)$. It is a powerful tool to describe uncertainty in parameter estimation. It can be calculated as:

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} \tag{1.5}$$

Given observed data D, the posterior distribution from the previous example can be calculated as:

$$p(y|x,D) = \int p(y|x,w)p(w|D)dw \tag{1.6}$$

Intuitively, it should be clear that this is the expectation over all possible models and it innately captures all the available information. However, calculating $p(w|D)$ is a difficult problem. Thus, we adopt an approximation approach and optimize a surrogate function. Bayesian learning is well-studied field. The methods for such approximations will be covered in a later section.

Decentralized Learning

Advances in machine learning and the paradigm shift in computing has brought a new wave of innovations that enable learning on decentralized data, namely decentralized learning. It includes all methods that enable learning on decentralized data that reside on participating devices. Participants that volunteer in the training of the model, do so by contributing both with their local data and with their computational resources. Their model is either partially or completely available to others directly or indirectly through other entities such as a central server. Moreover, participants may use a global model or their own private model. The training can be done either asynchronously or synchronously. To mathematically describe decentralized learning, let's denote the set of participants as P, models available at clients as $p(y|x;w_i)$, w_i defines the model of participant i, we are interested in approximating $p(y|x;\hat{w}_i)$ for each participant:

$$p(y|x;\hat{w}_i) \approx p(y|x;w_i), \forall\ i \in P \tag{1.7}$$

Decentralized learning serves as a fundamental starting point to utilize edge devices to train deep learning models, where data is produced and collected from. This would substantially decrease the communication costs, energy consumption at data centers, shrink the attack surface for data leaks, and pave the way for a privacy-preserving machine learning method that also emphasizes personalization. Moreover, it addresses some of the questions engendered by data collection, such as data ownership. In the next section, we explain the motivation behind decentralized learning further.

Motivation

Powerful models require mathematical rigor, tremendous amount of valuable data and an abundance of computational resources for training. However, if data at hand does not adequately represent the phenomena, then the performance of the model would suffer [6, 7, 8, 9]. The awakening of machine learning was in part engendered by the increase in the computational resources and the ability to make fast computations in parallel by utilizing GPUs[10]. Modern machine learning models are trained with data collected from millions of devices in large specialized data centers.

Many of the applications that make use of machine learning including computer vision, speech recognition, speech generation and text generation target the edge devices. Since edge devices, most importantly mobile phones, play a central role in our lives. These devices are used by billions of people and they include powerful sensors on-board. Naturally, they are also major data sources. They provide access to a large number of users with different usage patterns. The advances in mobile computing, specifically the advent and use of tensor processing units in mobile phones, enabled them to make inferences and execute training on-board. Moreover, efficient techniques and machine learning frameworks that provide a lighter versions for edge devices is gradually shifting the computation to edge devices[11]. Collectively, modern edge devices have both the data and the computational resources required to produce powerful models.

Today, technology monopolies are dominant in data collection and model training. They aggregate an unprecedented amount of data from their users and train machine learning models at gigantic data centers. Majority of the users are not sufficiently informed about the data collection and the technology that their data helps to train. This not only creates a privacy-risk for the users, but also brings about problems on data-ownership. The well-publicized data breaches has put the spotlight on data privacy and increased public concern on the topic. Moreover, only the most powerful institutions can gather the resources required for machine learning. Therefore, it is of utmost importance to provide the means of machine intelligence to public with a higher degree of freedom. Providing these means to the public would support the democratization process of machine learning. From the perspective of the companies, centralized learning approaches induce high infrastructure costs due to data collection and processing. Moreover, data collection brings about security risks that may negatively affect the company[12].

Decentralized learning is not only advantageous for the democratization process, but it has also been demonstrated that under certain conditions it provides better performance in terms of bandwidth and energy than centralized learning algorithms [13]. In addition to the communication costs, centralized learning processes are also retricted in terms of personalization. They provide one-for-all solutions to its clients. On the other hand, decentralized learning are inherently more suitable for personalization [14]. At the end of the training, models of $i \in P$ and

$j \in P$ potentially be different from each other:

$$w_i \neq w_j, \exists\, i \neq j \tag{1.8}$$

There are many orthogonal methods to achieve personalization after learning a common w^*, but it's also possible to achieve a degree of personalization during learning and use different objective functions for each participant. We will explore personalization more in section 3.1.

The setting of Decentralized Learning

Conventional machine learning algorithms collect data from different users into one large training dataset, where data is assumed to be i.i.d. Moreover, preprocessing techniques can be utilized to mitigate bias and redundancy. However, in decentralized learning, information on local datasets is limited and we have no control over the distribution. As mentioned previously, w is calculated by approximating $p(w|D)$. The foremost assumption of decentralized learning is the decentralized data, where all local datasets are a subset of D ($D_i \subseteq D$, $\forall\, i \in P$). How do we approximate $p(w|D)$, when training is executed at each device independently on their local datasets D_i. Specifically, how do we accumulate the knowledge acquired from $p(w|D_i)$? This approximation would yield a set of parameters $\hat{w}_i$ for participant i and we wish to encode the information of the whole dataset rather than its subset. The setting of decentralized learning includes many complex cases, where the answer should also satisfy the assumptions below.

Massively Distributed

Complex machine learning tasks require a large training dataset. Training samples can be distributed amongst millions or even billions of participant devices. Thus, decentralized learning should be scalable enough to support the accumulation of information from the local datasets. Moreover, it should provide a method to transfer the acquired knowledge, such that each client can achieve a certain degree of success on the tasks. Suppose there are P number of participants, the local dataset of participant i is denoted as D_i, then the total number of samples in the training dataset is $\sum_{i=1}^{P} |D_i|$ as the whole dataset is defined as the union of all local datasets:

$$D = \bigcup_{i=1}^{P} D_i \tag{1.9}$$

Since it is really hard to coordinate all the participants to train at the same time in such a massive graph, asynchronous decentralized learning approaches are more favorable. For example, the asynchronicity may be relaxed by allowing a subset of the participants to synchronize and train simultaneously. In [15], only a small subset of the *clients* are selected to be trained in parallel. If K is the number of clients trained at time t and P is the total number of clients, then $K \ll P$.

Non-Uniform Distribution (Highly Unbalanced)

The training samples may not be uniformly distributed amongst the participants. In fact, the chances are that the local datasets are non-uniform. It would be unrealistic to expect that the number of samples in the local datasets are equal, since dataset size will vary with respect to patterns and levels of usage. Moreover, it may be highly unabalanced. For example, some local datasets may contain thousands of samples, others might have less than ten samples. It depends on the usage pattern and frequency. The more the users interact with their devices, the more data will be aggregated. Denoting D_i as the local dataset of participant i and D_j as the local dataset of participant j, this inequality can be written as:

$$|D_i| \ll |D_j|, \forall\ i! = j \tag{1.10}$$

Non-IID Data Distribution

Since local datasets reflect the usage of each individual, they may represent a distribution far from the overall data distribution. Thus, data is likely to be distributed in a Non-IID fashion amongst the participants. For example, some labels might be missing from the local datasets, can be underrepresented or overrepresented.

Dynamic Data Availability (Time & Location)

Usage patterns vary with time and location. Thus, the local datasets would also reflect the time and location of the participating device. For example, samples drawn from participants at different times during the day will vary and samples drawn from participants in different countries represent the a distribution prevalent at that location. This adds another layer complexity to the training, as it also requires a degree of robustness against catastrophic forgetting.

$$D_i^t \neq D_i^{t+1} \tag{1.11}$$

Limited Connection

Edge devices are the biggest sources of data and some decentralized learning applications may target these devices. Edge devices lack the computational resources that complex machine learning algorithms require. Moreover, in some parts of the world, users have limited access to the internet with a constraint on their downloads or uploads. To encourage participation and collaboration, models should be optimized in terms of size and efficiency.

Unreliable Connection

Finally, participants may have unreliable connection. For example, users may voluntarily turn off their device during training or lose internet connection. We do not have direct control over the device, and all sorts of situations may arise that might result with a drop. Applications may have have users all around the world, including users from developing countries, where the internet service is not always available. The intermittent and unreliable internet service may severely affect training, destabilize the application or leak information. Thus, decentralized learning methods should be robust against these issues.

Graph Structure

The accumulation of the acquired knowledge depends on the graph structure. Decentralized approachefigss rely on a strongly connected graph structure, where information can be transferred from one node to another. Otherwise, it would not be possible to pass the accumulated knowledge to all participants. The graph structure may be dynamic or static, meaning that new nodes can be added to the graph or nodes can drop. However, here we are only interested in the learning of available information on the graph. Additionally, decentralized machine learning approaches should formalize the expected degree of success on different graph structures.

Hyperparameters

Hyperparameters are parameters that are set at the beginning of the training. As with all machine learning approaches, the optimal hyperparameters depend on the model and data distribution. Problems inherent to all machine learning approaches such as overfitting and exploding or vanishing gradients can highly affect the success rate of decentralized learning. Moreover, due to the collaborative nature of decentralized learning, there may be hyperparameters that control the collaboration of the users.For example, asynchronous methods may require training of multiple users in parallel, therefore the number of users to be trained simultaneously should be known beforehand. The hyperparameters may not be uniform, for example each user may train for a different number of local epochs. Hyperparameters can also be used to adjust the influence of users over training. Thus, it is vital to clarify the role of the hyperparameters and minimize the search space for optimal values.

Federated Learning

Federated learning is a decentralized learning method that enables asynchronous learning on a federation of clients coordinated by a central server. Its asynchronous nature provides a high level of flexibility. It has been shown that federated learning provides an overall accuracy

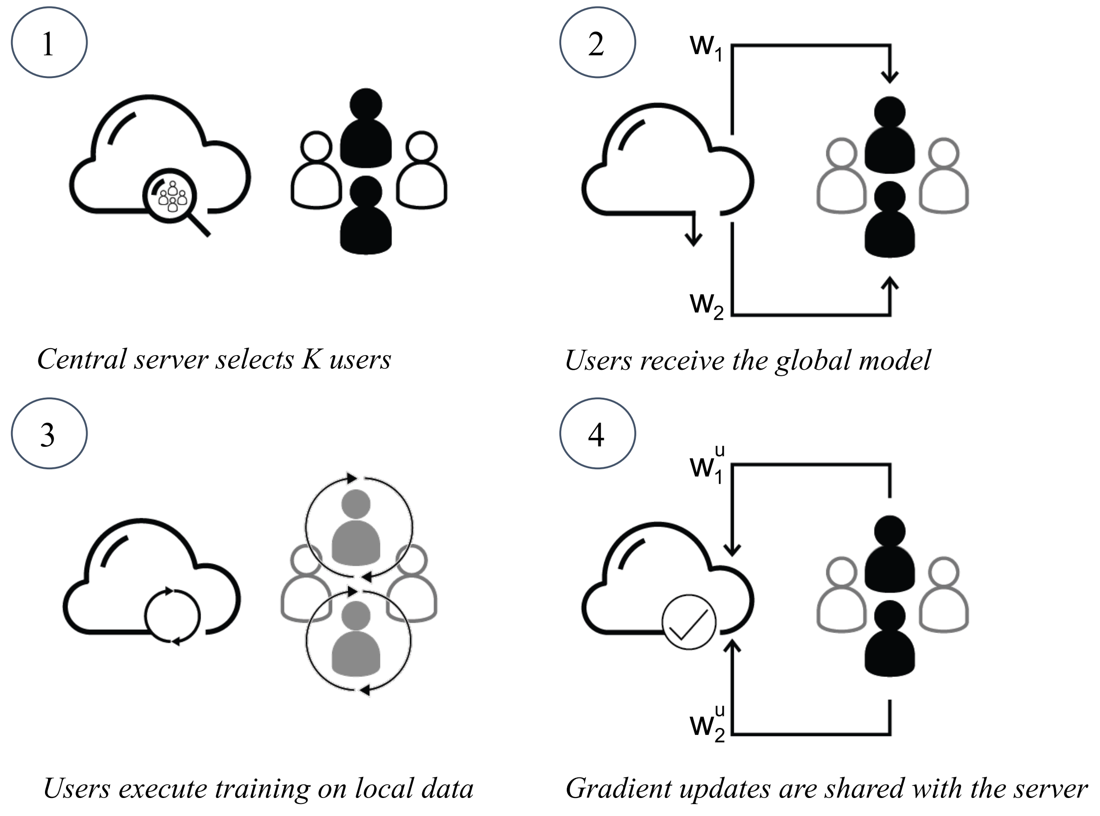

1
2
w1
w2
3
4
w1u
w2u
Central server selects K users
Users receive the global model
Users execute training on local data
Gradient updates are shared with the server

Figure 1.1. Federated learning procedure. (1) At the beginning of each communication round, central server selects a number of available users. (2) Users receive the current global model. (3) Users train the global model with their local data. (4) Gradient updates are sent back to the server, where the global model is updated by Federated Averaging.

comparable to the centralized machine learning methods[15].

During training, federated learning uses a *"star topology"*, where all the participants are connected to the central server as clients. At each communication round, the central server selects a number of available clients, and the training is executed on the selected clients. Federated learning utilizes stochastic gradient descent to optimize local models and the gradient updates are sent back to the central server. Then, the gradient updates are used to compute the new global model using a weighted average. The weights used in this process are defined as the local dataset size of the clients averaged by the total dataset size of the selected clients.

Differences between Federated Learning and Decentralized Learning

- Central Server (Coordinator): Federated learning is a case of decentralized learning, since there is a central server acting as a coordinator between the participants. Thus, federated learning is not fully-decentralized. Other decentralized learning approaches may or may not use such an entity.
- Global Model (Shared Initialization): At each communication round, federated learning uses a federation of participants to acquire new knowledge and the acquired knowledge is used to create a global model. The most recent global model is kept at the central server. Federated learning also assumes shared initialization. At the beginning of each communication round, central server sends the global model to the participants. The local model is initialized with the global model, and trained on the local dataset.

Neural Networks

Over the past decade, deep learning has become increasingly attractive in solving complex machine learning problems[16]. It's been shown to provide impressive results. Deep learning utilizes neural networks of different architectures to make inference and are often trained with efficient frameworks for tensor calculations. These architectures can be made up of several layers, where the intermediate layers between input and output layers are referred to as hidden layers. Each individual layer is constructed from a number of neurons, where the output depends on both the input, the parameters and a non-linear activation function. The input is processed with a linear function (an affine transformation) using a linear map and a translation term, before passing it on to the non-linear activation function. For example, to express the operations on a fully-connected layer of a feed-forward network, let's denote W as the weight matrix used to transform an input vector x, b as the bias term, σ as the non-linear activation function, then the output vector y given x is:

$$y = \sigma(Wx^T + b) \tag{1.12}$$

The neural network reaches an output after passing the input from the layers iteratively and gradually transforming it to an output. In other words, the layers are used to apply a mapping and the decision is made by utilizing this mapping.

During training, deep neural networks inscribe the information on the data to its structures using the parameters (w) and generate a mapping. Even in the deepest neural networks, the earlier layers is encoded an information on the data distribution. This is achieved by backpropagation. After loss is calculated, the gradients are calculated starting from the output layer and backward propagated through the model until it reaches the input layer. Deep neural network architectures can be improved and extended to enhance model expressiveness and encode the information better. As the number of parameters increase in the model, it allows the model to capture more information on the training dataset. However, it becomes harder to train and more prone to overfitting. Thus, regularization methods are used during training as a precaution and to boost the generalization of the model. During training, overfitting can be recognized from the comparison of training and validation losses. It often manifests itself as an increasing validation loss, while training loss is stably decreasing or has converged. Overparametrization of neural networks is an example to the structural uncertainty that arises when constructing neural networks, which is well represented in the bayesian context.The uncertainty in deep learning models is a well studied field [17, 18, 19].

Parameter Estimation

There are various methods to estimate the parameters of a neural network. Differential methods are used for this approximation including stochastic gradient descent (SGD) and Adam optimizer [20]. We had previously mentioned bayesian approach to parameter estimation. Let's also briefly visit a frequentist approach to parameter estimation, before we move onto the optimization methods.

Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) does not convey prior knowledge to the model. It only uses the training dataset. For a training dataset $D = \{(y_1, x_1), (y_2, x_2) \cdots, (y_N, x_N)\}$, it can be written as:

$$w \sim \arg\max_{w \in W} p(D|w) \tag{1.13}$$

$$w \sim \arg\max_{w \in W} \prod_{i=1}^{N} p(y_i|x_i, w) \tag{1.14}$$

$$w \sim \arg\max_{w \in W} \sum_{i=1}^{N} logp(y_i|x_i, w) \tag{1.15}$$

In other words, we choose the values that maximize the likelihood distribution, $p(D|w)$. Moreover, we keep the parameters as values rather than a distribution on the parameters. For example, for a coin toss, the frequentist approach would be to count and calculate the frequencies of heads and tails. During inference, a prediction is made using this frequency. We will use the term vanilla neural network, whenever the parameters are calculated by maximum likelihood estimation and bayesian neural network whenever the parameters are calculated by bayesian parameter estimation.

Gradient Descent

During training, we want to minimize the loss given the whole dataset, D. The derivative of the loss function with respect to the parameters point to the steepest ascent. Thus, we want to go in the negative direction of the gradient. However, the gradients do not give the distance to the point where loss is minimum. They only give a direction. Therefore, we update the parameters, after gradients are scaled with the learning rate, η:

$$w = w - \eta \times \frac{\partial L(D)}{\partial w} \tag{1.16}$$

Here the tuning of learning rate plays an important role for optimization, because if it's too small, we might get stuck in a local minima far from the minimum. If it's too large, then it might jump over a minimum. Moreover, this optimization procedure is an expensive operation. Thus, we randomly select mini-batches of K samples from the training dataset, and optimize the parameters with every mini-batch. The use of mini-batches increase the speed of optimization and the stochasticity of this process provides an unbiased estimation of the true gradient. This optimization method is called mini-batch gradient descent. In practice, gradients are calculated by automatic differentiation tools and the optimization process is efficiently automated.

Chapter 2

Bayesian Learning

Modelling Uncertainty

Uncertainty plays a fundamental role in decision making and it emerges mainly from three situations; uncertainty introduced by noisy data, uncertainty on the optimal model parameters and uncertainty on the model structure.

Uncertainty Introduced by Data

In machine learning, models trained on observed data are often used to make predictions given unobserved data[21], but how can we trust the observations? Measurements can be noisy due to sensory error, such as noise introduced by a defective microphone or blurry images from a damaged camera. Therefore, there is an intrinsic uncertainty that starts with measurement and data acquisition. Given that neural networks have poor confidence calibration[22], this can have dire consequences. Imagine a scenario, where an autonomous vehicle makes an overconfident decision about the proximity of another vehicle due to a sensory error. If the driver is not aware of the situation, this may result with a fatal traffic accident. However, if we can successfuly convey this uncertainty to the model and calibrate the predictions accordingly, the driver can be notified to take over the control before it is too late.

Uncertainty about Optimal Model Parameters

The goal of training is to learn a model rather than find the optimal values for individual parameters[23]. In very large models we do not know the role individual parameters play in a prediction. Therefore, it is not known whether a given set of parameters is optimal for predicting unobserved data. In practice, datasets are often limited and we may unknowingly push our model to overfit the dataset. Then, how do we adjust individual parameters and avoid situations like overfitting? Regularizers such as weight decay are often used to prevent overfitting as they give a degree of control over model complexity[24, 25]. It has also been demonstrated that they play an ameliorative role in model calibration[22]. However, by themselves regularizers do not fully convey the uncertainty on the individual parameters to the model.

Uncertainty on the Model Structure

There are many possible models for any given task. It might not be certain whether support vector machines or neural networks would be optimal[26, 27]. Furthermore, the optimal size of a model is unknown[28]. If the model has too many parameters for the task, it would overfit the data. In order to prevent overfitting, we may want to limit the number of parameters. However, if it is oversimplified, then the model would not be able reach our expectation and might even perform worse.

Model Uncertainty in Decentralized Learning

Until now, we considered the motivation behind uncertainty modelling from a centralized machine learning perspective. Now, let's consider this topic from a decentralized perspective. The uncertainty in the centralized setting is also relevant for decentralized machine learning. The model on each participant is affected by the models on other devices. This increases data uncertainty as a number of participants might contribute to the global state of the network with knowledge acquired from faulty data. Moreover, decentralized learning must enable each participant to internalize the knowledge acquired on other devices. Given that personalization is a vital aspect of decentralized learning, from the perspective of an individual participant there is an uncertainty over the true values of the parameters. Finally, when the participants train with their own data, the information on data observed by other participants should not be lost. To the contrary, individual contributions should be accumulated.

Bayesian Perspective on Uncertainty

The space of exploration is too large to search for the optimal probability distribution and consider all the uncertainties. Therefore, they should be conveyed to model to make more well-informed decisions. In the bayesian frontier, one view suggests to find the probability of several different model sizes given data and use it when making predictions [23]. Suppose that M_i represents an arbitrary model from a set of selected models appropriate for a task then:

$$p(M_i|D) = \frac{P(D|M_i)P(M_i)}{P(D)} \tag{2.1}$$

$$p(D|M_i) = \int p(D|M_i, w)P(w|M_i)dw \tag{2.2}$$

A second more prevalent view follows a prior-based approach, and argues that constraining model-size is unnecessary as long as the prior-belief conveys the uncertainty to the model. Thus, the number of parameters in a model should not be limited by Occam's Razor or using the limited observed data. Bayesian learning leads to an automatic Occam's Razor and regulates the model in terms of complexity with the prior. Moreover, it has consistently been shown that Bayesian neural networks can reach the performance of well-regulated ordinary neural networks[29, 30]. In practice, using smaller Bayesian models are better as long as a similar performance is achieved due to faster computations and the smaller memory allocation. This is especially important for decentralized learning problems, where users usually have constrained resources.

Bayesian Parameter Estimation

Bayesian parameter estimation inherently provides a framework to formalize this uncertainty and the ability to incorporate all available knowledge to the model. In Bayesian statistics, posterior distribution gives context to the decision-making process. It is calculated by marginalising over the parameters as in equation 1.6. However, calculating $p(y|x)$ is not straightforward and is computationally intractable for high-dimensional problems. Since it can not be solved in polynomial time, we adopt an approximation approach. In other words, we try to answer queries about the original distribution by simulating it. In the rest of this section, we will explore the approximation methods namely, markov chain monte carlo sampling and variational inference.

Markov Chain Monte Carlo

For a long time, Markov Chain Monte Carlo (MCMC) methods were the preferred method for approximating the posterior. MCMC methods that are closely related to our topic include; Metropolis-Hastings sampling[31, 32, 33], Gibbs sampling (a special case of Metropolis-Hastings) and Hybrid Monte Carlo (HMC)[18] methods. Although these methods are beyond the scope of this work, it is important to have a general understanding of MCMC methods.

In machine learning and statistics, the widespread use of MCMC started in the early 90's[34], where in 1995 Raymond Neal pioneered its use in Bayesian Neural Networks[35]. Instead of calculating the integral, MCMC methods generate random samples from the posterior distribution and thereby simulate the posterior. After generating a large enough sample set, the mean, variance or any other characteristic for the distribution can be estimated. The degree of accuracy in this estimation can be adjusted as desired. As the number of samples grow, the accuracy of the estimation increases. The reasoning behind MCMC can be induced from the fact that the characteristics of a distribution can be empirically estimated from the samples drawn from it. For example, given Gaussian distribution $N(\mu, \sigma)$, μ can be estimated from the random variable $x \sim N(\mu, \sigma)$, since:

$$\mu = E[x] = \int_{-\infty}^{\infty} x f(x) dx \tag{2.3}$$
$$= \lim_{s \to \infty} \frac{1}{s} \sum_{i=1}^{s} x_i \tag{2.4}$$

MCMC methods generate samples by fixing the remaining variables and previously drawn samples. Let's give a brief example to this "sweeping" behavior from Gibbs sampling to clarify it further[36]. Suppose we have a pair of random variables (X,Y), we first sample from a prior $x^0 \sim q(x)$, then we alternately sample from the distributions as follows:

$$\begin{aligned}
y^0 &\sim p(y|X = x^0) \\
x^1 &\sim p(x|Y = y^0) \\
y^1 &\sim p(y|X = x^1) \\
&\vdots \\
y^{i-1} &\sim p(y|x^{i-1}) \\
x^i &\sim p(x|y^{i-1}) \\
&\vdots \\
x^s &\sim p(x|y^{s-1}) \\
y^s &\sim p(y|x^s)
\end{aligned}$$

The sequence of random variables generated by this process is called *"Gibbs sequence"*: $x^0, y^0, x^1, y^1, x^2, \cdots, y^{i-1}, x^i, \cdots, x^s, y^s$. Using the Gibbs sequence with a large enough s, we can get approximations close to the true distribution. However, it is particularly hard to successfully approximate the true distribution for large datasets and complex models. It has been shown that the convergence of MCMC methods require all parameters to have converged [37], which portends bad convergence rates for models such as neural networks. Given the size of the modern datasets and complexity of models such as deep neural networks, *variational inference* has been adopted as the primary approximation method.

Variational Inference

The fundamental idea behind variational inference is to use a proxy distribution to approximate the posterior. This is achieved by iterative optimizations over the latent variables of this proxy distribution. In other words, we search over the space of approximating distributions and find the distribution that is closest to the posterior. Since it follows an optimization approach, it allows the use of stochastic optimization and distributed optimization methods. Therefore, variational inference scales more easily to large datasets compared to MCMC methods. This does not mean that variational inference is always better than MCMC methods, as they both enjoy some theoretical guarantees. Although it is known that variational inference understates the variance of the posterior density as a result of its objective function[38], it has also been shown that this does not translate to accuracy loss [39]. Given that scalability is what essentially matters for decentralized learning, in this work we stick to variational inference.

To find the distribution that *closely* approximates the posterior, we want to minimize a measure of distance between the proxy and the posterior. Particularly, we find the distribution that minimizes the Kullback-Leibler (KL) divergence. Following the classification example from the previous chapter, let's explore the theory behind variational inference. Assume that the training dataset D is a collection of observations and corresponding labels $D = (x_0, y_0), (x_1, y_1)..., (x_n, y_n)$. There are n number of samples in D. When making a prediction $\hat{y}$ given a test observation $\hat{x}$, we try to minimize the expected loss $E_{p(w|D)}[p(\hat{y}|\hat{x}, w)]$. This translates to $\int p(\hat{y}|\hat{x}, w)p(w|D)dw$, and we had previously defined $p(w|D)$ as:

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} \tag{2.5}$$

$p(D)$ is unfortunately intractable since it requires the calculation:

$$p(D) = \int p(w, D) dw \tag{2.6}$$

Thus, rather than compute $p(D)$, we approximate $p(w|D)$ using a proxy distribution $q(w|\theta)$, where the distribution $q(.)$ is parametrized by the latent variables θ.

$$q^*(w|\theta) = \underset{\theta \in \Theta}{\arg\min} KL(q(w|\theta)||p(w|D)) \tag{2.7}$$

$$q^*(w|\theta) \approx p(w|D) \tag{2.8}$$

After finding the set of latent variables that minimizes KL-divergence between the proxy and the posterior, we find the expected loss as $E_{q^*(w|\theta)}[p(\hat{y}|\hat{x}, w)]$. $KL(q(w|\theta)||p(w|D))$ can be calculated as:

$$KL(q(w|\theta)||p(w|D)) = \int q(w|\theta) log \frac{q(w|\theta)}{p(w|D)} dw \tag{2.9}$$

$$= \int q(w|\theta) log \frac{q(w|\theta)}{p(w)p(D|w)} dw + logp(D) \tag{2.10}$$

With this factorization, KL divergence also yields a constant term, which is also called the log evidence $logp(D)$. Let's note that it is a non-negative value. Moreover, since it is a constant term, if only consider the first term during minimization. Thus, the equivalent objective function is:

$$\underset{\theta \in \Theta}{\arg\min} \int q(w|\theta) log \frac{q(w|\theta)}{p(w)} dw - \int q(w|\theta) logp(D|w) dw \tag{2.11}$$

$$\underset{\theta \in \Theta}{\arg\min} \; KL[q(w|\theta)||p(w)] - E_{q(w|\theta)}[logp(D|w)] \tag{2.12}$$

Let's remember that the factorization of the objective function also yields the log evidence and that KL divergence has a non-negative value. Therefore, minimizing KL divergence is equivalent to maximizing the evidence lower bound (ELBO).

$$\underset{\theta \in \Theta}{\arg\max}\ E_{q(w|\theta)}[logp(D|w)] - KL[q(w|\theta)||p(w)] \tag{2.13}$$

$$logp(D) \geq E_{q(w|\theta)}[logp(D|w)] - KL[q(w|\theta)||p(w)] \tag{2.14}$$

This objective function also outlines a trade-off between the log-likelihood and the prior. The intuitive explanation is that maximizing $E_{q(w|\theta)}[logp(D|w)]$ forces the $q(w|\theta)$ to explain the observed data well, while minimizing $KL[q(w|\theta)||p(w)]$ increases the similarity between $q(w|\theta)$ and the prior $p(w)$.

Decentralized Learning & Bayesian Parameter Estimation

Bayesian parameter estimation aims to incorporate all available knowledge to the model. We have explored how it is achieved locally in centralized learning, but what does *”all available knowledge”* mean in the context of bayesian neural networks? In centralized learning, *”all available knowledge”* includes not only the uncertainties and observations, but also initial beliefs. One can use informative priors to reflect the expert’s domain knowledge to the model. In the context of decentralized learning, we can reinterpret the prior to include the accumulated knowledge.

Priors are used to capture the prior beliefs and the process of learning starts with the priors. Before any data is observed, the model would only have its prior. When the model starts observing data, its learning would be built upon its prior beliefs and we update this to a posterior distribution. After observing sufficient amount of data, model would depend more on its observations. Its own experiences become more important in its decisions than its prior beliefs.

In the context of collective intelligence, priors may be used to capture the previously acquired knowledge. Let’s use an analogy to illustrate the utility of priors and prior beliefs. We may not have experienced something on our own, but fortunately as humans we are capable of using other's experiences in our decision making. Before we experience a phenomena on our own, our decisions would reflect our prior beliefs. As we get more experienced, we would rely on our own experience rather than others.

Prior-based approaches to decentralized learning build upon prior beliefs and take advantage of the priors to accumulate knowledge. Suppose a new node joins to a decentralized learning graph. The fledgling node has no data available. Fortunately, it is connected to the graph. Its neighbors can communicate parts of their model with it. By doing so, they would also be passing the accumulated knowledge. At the beginning, the node's decisions would rely more on other's observations than its own. During learning, its model would be built on the global state of the graph. Then, it may choose to broadcast its beliefs and acquired knowledge to its neighbors. The node's publicized beliefs would be ”absorbed” by the graph. This can be generalized to all nodes. Learning in decentralized learning builds on the global state, which can be conveyed to the nodes as prior belief.

Bayesian Neural Networks

The bayesian approach to neural networks conveys the aforementioned uncertainties to the model. Research on bayesian neural networks build on the works of David MacKay[17] and Radford M. Neal [18]. The intractabilities of bayesian learning is also inherent to bayesian neural networks. Because of the complexities of neural networks, variational inference is often the method of choice. Although variational inference renders the calculation of the posterior to an approximation, the calculation of the expected log-likelihood remains to be impractical. Therefore, once again we will have to approximate the expectation.

$$-logp(D|w) \approx \frac{1}{S}\sum_{k=1}^{S} -logp(D|w_k = q_k(\theta)), \tag{2.15}$$

where w_k is the k^{th} Monte Carlo sample drawn from a distribution $q(\theta)$ and S is the number of samples drawn. This would still require a lot of computation if we need to sample S times for each observation $(x, y) \in D$. Fortunately, it has been shown that variational inference is amenable to mini-batch gradient descent [40]. A more recent work approaches to the problem from a more practical perspective [30] and improves upon [40]. In particular, [30] utilizes reparametrisation on the expected log-likelihood MC estimates. All implementations in this work follow [30], which was subjectively the simplest and most practical approach. The fundamental idea behind this approach is to use the pathwise derivative estimator rather than the characteristic function estimator [19]. Gradients can then be calculated simply by backpropagation and the parameters can be updated with gradient descent. Reparametrization trick specifically recasts the weights as $q(\theta, \varepsilon)$, where ε is a random variable. In our implementations, we use a Gaussian distribution parametrized by $\theta = (\mu, \sigma)$ for the variational posterior $q(w|\theta)$. Then, using $\varepsilon \sim N(0, 1)$, we can sample the weights as $w = \mu + \sigma\varepsilon$. Furthermore, we reparametrize σ as $log(1 + exp(\rho))$, which always yields a non-negative value.

Local Reparametrization Trick

As previously mentioned, variational inference understates the variation of the posterior. Fortunately, variance of the estimator can be further minimized by employing the local reparametrization trick [41], which samples from the resulting activations rather than the weights. The intuition here is as follows. The weights affect the expected log-likelihood only through the activations. For a posterior, whose weights are parametrized by a Gaussian distribution, the resulting activations would also be a Gaussian. Monte Carlo estimators accuracy can be adjusted with the number of samples that it generates. When we sample from the weights to calculate the expected log-likelihood, the variance of the estimator is inversely proportinal to the number of the weights. The reparametrization allows us to sample from the activations. Given that the number of activation variables are less than the number of the weights, the number of samples required to be generated is lower.

Implementation

In this work, we follow the implementation introduced by Blundell [30]. Thereby, the objective function $F(D,\theta) \approx KL[q(w|\theta)||p(w|D)]$ is:

$$F(D,\theta) \approx \sum_{s=1}^{S} logq(w_s|\theta) - logp(w_s) - logp(D|w_s), \tag{2.16}$$

where w_s is the s^{th} sample generated and S is the total number of Monte Carlo samples. We also draw comparisons between closed form and open form KL divergence calculations with respect to performance. The objective function of the closed form KL divergence calculation translates to:

$$F(D_i,\theta) \approx \frac{1}{M} KL[q(w|\theta)||p(w)] + \frac{1}{S}\sum_{k=1}^{S} -logp(D_i|w_k), \tag{2.17}$$

Since weights of the variational posterior are sampled from a diagonal Gaussian distribution, if the prior is also a Gaussian KL-divergence can be calculated in closed-form. Let prior distribution ($p(w|\alpha)$) be a Gaussian with mean μ_α and variance σ_α, then the closed form KL divergence can be calculated as follows:

$$KL[q(w|\theta)||p(w;\alpha)] = \frac{1}{2}[log\frac{|\Sigma_\alpha|}{|\Sigma_\theta|} - d + tr(\Sigma_\alpha^{-1}\Sigma_\theta) + (\mu_\alpha - \mu_\theta)^T \Sigma_\alpha^{-1}(\mu_\alpha - \mu_\theta)] \tag{2.18}$$

where $\theta = (\mu_\theta, \sigma_\theta)$ and $\alpha = (\mu_\alpha, \sigma_\alpha)$. Notice that KL divergence is calculated once for each mini-batch instead of each w_k sample. The detailed procedure for all implementations are given below.

Table 2.1. Notation for Algorithm 1 & 2.

Algorithm Notation	
N	Number of samples in one mini-batch
M	Number of mini-batches per epoch
S	Number of times weights are sampled; Monte Carlo Estimator
D	Training dataset
NOTE: w_k is k^{th} Monte Carlo sample drawn from the variational posterior	
NOTE: $D_{(m)}$ is m^{th} mini-batch	
NOTE: $\theta = (\mu, \rho)$	

Algorithm 1. Bayes by Backprop. KL Divergence calculted from log probabilities.

```
for $i = 1$ to $M$ do:
    for $k = 1$ to $S$ do:
        $\varepsilon_k \sim N(0, I)$
        $w_k \leftarrow \mu + \log(1 + \exp(\rho)) \circ \varepsilon_k$
        $\log q(w_k|\theta), \log p(w_k;\alpha) \leftarrow$ Calculate log probabilities
        $\log P(D_i|w_k) \leftarrow$ Perform forward pass over the mini-batch
    end for
    $F(D_i, \theta) \leftarrow \frac{1}{S}\sum_{k=1}^{S}\frac{1}{M}[\log q(w_k|\theta) - \log p(w_k;\alpha)] + \frac{1}{S}\sum_{k=1}^{S} - \log p(D_i|w_k)$
    $\theta \leftarrow \theta + \eta \nabla_\theta F(D_i, \theta))$
end for
```

Algorithm 2. Bayes by Backprop. KL Divergence calculted in closed form.

```
for $i = 1$ to $M$ do:
    for $k = 1$ to $S$ do:
        $\varepsilon_k \sim N(0, I)$
        $w_k \leftarrow \mu + \log(1 + \exp(\rho)) \circ \varepsilon_k$
        $\log P(D_i|w_k) \leftarrow$ Perform forward pass over the mini-batch
    end for
    $F(D_i, \theta) \leftarrow \frac{1}{M}KL[q(w|\theta)||p(w;\alpha)] + \frac{1}{S}\sum_{k=1}^{S} - \log p(D_i|w_k)$
    $\theta \leftarrow \theta + \eta \nabla_\theta F(D_i, \theta))$
end for
```

Chapter 3

Decentralized Learning

Introduction

Decentralized learning attracted tremendous attention to itself in the last few years. It is a promising field in machine learning that enables learning on decentralized data. Particularly, because it does not require data to be transmitted to an external server. Instead it requires only the transmission of acquired knowledge. It shrinks the surface for privacy risks. Therefore, it alleviates the problems engendered by data-privacy and data-ownership. It creates a framework for which data-privacy can even further minimize the risks involved in knowledge sharing. Moreover, it introduces a degree of personalization, whereas centralized learning methods provide a one-for-all model to its users. For example, adapting the model to the individual usage-patterns would increase the accuracy of recommender systems.

A more prominent application for decentralized learning would be in healthcare applications, where data-privacy is of utmost importance. Using decentralized learning in these applications would not only mitigate the privacy-risks, but also facilitate a degree of personalization for personalized care. Of course, the decisions would not only rely on the data provided by the individual. It would also depend on the knowledge acquired from others.

The advantages of decentralized learning are not limited to personalization and data-privacy. The centralized approaches require data to be collected in the central server. Often, data is too large to be processed and has to be distributed. Therefore, the central server redistributes data to other entities with adequate computational resources, where learning actually takes place. However, this approach to learning creates a communication bottleneck and slows down learning [13]. On the other hand, decentralized learning enjoys a relative freedom with respect to communication costs. Moreover, decentralized learning can be easily scaled up. It has been shown that more nodes can increase the speed of learning [13].

Due to the advances in mobile computing, we are witnessing a paradigm shift in computing. Mobile devices have become powerful enough to run complex models and the new trend is to move computations to the ”client-side”. This is mostly incentivized from high infrastructure costs induced by centralized approaches. In parallel, public concern on data-privacy increased significantly, urging companies to take more precautions. Research on decentralized learning has gained pace to provide a framework to achieve training independently and directly on the data sources. There are two main approaches to decentralized learning: *fully decentralized learning* and *federated learning*. Federated learning uses a central server to coordinate learning. At each communication round, central server selects a subset of clients and sends them the global model. Clients train the current global model with their local data and send the gradient updates. The central server aggregates the gradient updates and computes the new global model. It is called federated learning, because the learning is achieved asynchronously by a federation of clients. Nevertheless, all this coordination creates traffic on the central server. Therefore, federated learning is also subject to the aforementioned communication bottleneck. Fully-decentralized learning enjoys a relative freedom, as all the calculations on the nodes are achieved independently at all nodes and nodes communicate the acquired knowledge to their one-hop neighbors. The sparsity of the communication graph enables ease of training in terms of communication traffic.

Background Work

Research on decentralized optimization is rooted in the seminal works published in 1980s [42, 43]. Decentralized optimization has applications in various areas including opinion dynamics analysis, network learning, cooperative robotics, communication networks as well as sensor networks. Decentralized learning approaches the *empirical risk minimization* problem from a decentralized optimization perspective and find an optimal set of parameters over the whole dataset. Many methods also try to minimize the communication costs. Significant progress has been made in the field, but there is still room for improvement.

In essence, empirical risk minimization problem is minimization of the expected loss over the training dataset:

$$w^* = \arg\min_{w \in R^d} L(w), \; where \; L(w) = E_D[l(w)] = \frac{1}{n}\sum_{i=1}^{n} l_i(w), \tag{3.1}$$

$l_i(.)$ is the loss function that calculates the loss of w given datapoint $(x_i, y_i) \in D$ and n is the number of datapoints in the training dataset. We try to minimize the total loss over all datapoints in D through optimizing w. The decentralized learning objective function can be written as:

$$w^* = \arg\min_{w \in R^d} L(w), \; where \; L(w) = \sum_{k=1}^{K} \frac{n_k}{n} L_k(w), \tag{3.2}$$

$$L_k(w) = \frac{1}{n_k}\sum_{i=1}^{n_k} l_i(w), \tag{3.3}$$

where K is the number of nodes and n_k is the number of samples in the local dataset of node k. L_k is the expected loss given the local dataset at node k. If data was i.i.d amongst the nodes, then the expected loss would be the same at all the nodes, $E[L_i(w)] = E[L_j(w)], \forall i, j \in P$, and the problem could essentially be solved like a centralized learning problem, since:

$$w^* = \arg\min_{w \in R^d} \frac{1}{K}\sum_{k=1}^{K} E[L_k(w)] \tag{3.4}$$

Unfortunately, data can be highly-unbalanced and non-iid in decentralized learning. Therefore, we can not make such assumptions. Moreover, solving this problem is comparatively harder for neural networks because of its size and complexity. In section 1.7, we explained how gradient descent and stochastic gradient descent was applied in a centralized setting. Over the past few years, federated optimization methods that combine the benefits of both GD and SGD have been proposed [44]. However, federated optimization uses a central entity to coordinate training, and keeps the most recent global model. Fully-decentralized optimization is a more demanding problem, since learning is achieved with only the locally available information and the communication between the immediate neighbors.

Gossip Algorithms

Gossip algorithms is an important family of algorithms, which has a problem definition structurally close to empirical risk minimization. Gossip algorithms try to answer consensus problems such as decentralized averaging [45, 46, 47], where the computational costs are distributed amongst the nodes and nodes communicate with their neighbors to find a solution. Moreover, gossip algorithms do not require synchronization between nodes. Note that empirical risk minimization and decentralized averaging problems are both structured as a finite-sum. The mean of data distributed amongst K nodes is:

$$x_{avg} = \frac{1}{K} \sum_{k=1}^{K} x_k, \tag{3.5}$$

where x_k is the datapoint on node k. [48, 49] consider a problem that includes non-smooth functions and propose subgradient methods to address the problem.

Parallel Stochastic Gradient Descent Variants

The main idea behind parallel stochastic gradient descent (PSGD) variants is to use SGD to train on local data in parallel and aggregate the acquired knowledge. In an early work on the topic, the aggregation step is achieved by simply taking the average of local updates to get a solution w^* [50]. The synchronous and asynchronous decentralized parallel stochastic gradient descent algorithms were analyzed by several works [51, 52, 53]. PSGD variants have also been widely studied to solve large-scale machine learning problems in deep learning [54, 55]. The federated averaging method improves upon EASGD and adapts it to the federated learning setting [15]. LEASGD builds upon EASGD to provide a differentially-private method with reduced communication costs [56]. Another recent work proposes a new algorithm D^2 that is robust to data variation among the nodes [57]. An overview of PSGD algorithms, including a extensive comparison between the Centralized-PSGD (C-PSGD) and Decentralized-PSGD (D-PSGD) is provided in [13]. Moreover, D-PSGD algorithms are demonstrated to perform better than C-PSGD algorithm under certain conditions.

[58] analyzes the convergence of decentralized gradient descent approach to the consensus problem. [59] presents EXTRA which uses a gradient tracking technique. It is the first modified version of decentralized gradient descent that uses fixed-step size. Hence, it performs significantly better than decentralized gradient descent. Another solution proposed in [60] builds on previous stochastic incremental averaging gradient algorithms; SAG algorithm [61] and its unbiased version *SAGA* [62]. However, SAG and SAGA is not very suitable for neural networks because of their memory requirements. Lastly, [63] extends a distributed optimization method CoCoA to the decentralized setting and provide significant improvements over previous methods for learning linear models [64]. Decentralized optimization is a field that has received significant attention from the machine learning community as well. [65] provides an overview of methods from a machine learning perspective.

Federated Optimization

Federated optimization is similar to distributed optimization methods with more restrictive assumptions. The fundamental idea of federated optimization is that it requires a parameter server that is shared by all the nodes. Rather than follow the decentralized optimization approaches explained in the previous section, the centralized distributed optimization methods are extended to the federated learning setting [44]. Research on federated optimization is aligned with this very idea and focus on communication cost optimization under stricter assumptions on data distribution. One such problem is distributed mean calculation without assuming that data is i.i.d.. Distributed mean calculation is a vital step in federated averaging, since the central server computes a weighted average of the model updates to create the new global model at the end of each communication round. This problem was most recently addressed with a communication efficient solution in [66]. Since federated optimization is interested in solving optimization problems in the federated learning setting, data is assumed to be massively distributed, highly unbalanced and non-iid. The prob lem definition of federated optimization is clarified and relevant optimization algorithms are reviewed by Konecny in [44]. Furthermore, Federated SVRG (FSVRG) algorithm is proposed as a solution to the federated optimization problem. FSVRG is a combination of DANE and SVRG algorithms [67, 68]. The main idea of FSVRG is to utilize the quadratic perturbation trick of DANE and use SVRG to skip the expensive operations inherent in DANE. More specifically, SVRG is used to calculate a stochastic update rather than the exact minimum required by DANE. An extensive literature review on stochastic, distributed and federated optimization methods can be found in [69].

Another line of work follow alternating direction method of multipliers (ADMM) [70] for distributed optimization, which is well-suited to solve distributed convex optimization problems with a central entity. [71] improves upon ADMM and modifies it for larger networks. [72] proposes a broadcast-based method that enables agents to collectively solve the minimization problem. [73] improves upon previous work and proposes a method that enables asynchronous updates.

Personalized Optimization

The first significant step towards fully decentralized learning in the context of personalized models was made by Vanheasebrouck [14], which builds upon the gossip algorithms literature. However, rather than learning a global objective, nodes define their own objective and train accordingly. Since nodes define their own objectives, individual models are learnt as:

$$\theta_i^{ml} \in \underset{\theta \in \Theta}{\arg\min}\, L(\theta, D_i), \tag{3.6}$$

where θ_i^{ml} is the learned model at node i. The network graph that defines the nodes and their relations is assumed to be given as a weight matrix W and a set of edges E. The weights are assumed to reflect task similarity between the nodes. Nodes are only aware of their one-hop neighbors. The two asynchronous methods proposed in this work are *model propagation* and *collaborative learning*. Throughout learning, both methods seek an adjustable degree of smoothness over the network graph. Each node holds a confidence value on its training. By weighting the contributions of the node's local training and belief over its neighbors' models to the objective function, the confidence levels can be controlled and reflected to the training.

While model propagation achieves learning in two phases, collaborative learning interweaves local learning and regularization using Alternating Direction Method of Multipliers (ADMM) [70]. Nodes update their beliefs over their neighbors' models at each communication round. The objective function of model propagation is formalized as:

$$F_{MP}(\Theta) = \frac{1}{2}\Big(\sum_{i<j}^{n} W_{ij}||\theta_i - \theta_j||^2 + \mu \sum_{i=1} D_{ii}c_i||\theta_i - \theta_i^{ml}||^2\Big), \tag{3.7}$$

where $D_{ii} = \prod_{j=1}^{n} W_{ij}$ is used as a normalization term, $\mu > 0$ is the trade-off parameter, and c_i is the node's confidence on its local training. Note that this formalization carries the weighting and thereby the trade-off inherent to decentralized learning. Minimizing the first term would make the model more similar to the neighbors' model, while minimizing the second term would make the model to rely more on local data. By generalizing a semi-supervised label propagation technique [74], an asynchronous algorithm is provided as a solution. Instead of learning the subproblems locally and then propagating the model, collaborative learning approach enables the nodes to learn simultaneously using both their local datasets and the behavior of their neighbors.

The objective function of collaborative learning is defined as:

$$F_{CL}(\Theta) = \sum_{i<j}^{n} W_{ij}||\theta_i - \theta_j||^2 + \mu \sum_{i=1}^{n} D_{ii} L_i(\theta_i) \tag{3.8}$$

Notice that the first term is the same as before, enforcing a degree of similarity between a node and its neighbors. However, the second term is changed and alleviates the accuracy decrease on the local dataset. Authors reformulate the problem as a partial consensus problem and solve it using a decentralized gossip algorithm. Bellet [75] builds on collaborative learning method and uses a block coordinate ascent algorithm instead of ADMM to solve the objective function of collaborative learning method. The proposed solution does not require any auxiliary variables. Moreover, since it's based on block gradient descent rather than ADMM, nodes can broadcast their models to their neighbors at once. Thus, it significantly outperforms the previously proposed ADMM-based algorithm. A recent work recasts the problem as a multi-task learning problem [76]. Authors enable decentralized learning of personalized non-linear classifiers with their novel reformulation based on l_1-Adaboost [77] and present an asynchronous and decentralized solution that relies on peer-sampling service [78]. With a combination of base predictors, authors depend on $l1$-Adaboost's ability to build non-linear classifiers. This ability separates their method from previous works.

Peer-to-Peer Learning

Decentralized learning can be cast as a problem of social learning, since we try to optimize models by training only on the observations residing on independent devices. The observerations reflect usage patterns, users and their interactions. With a continuous flow of observations, more iterations and communication between the devices, models would would eventually get closer (if a consensus is sought). This is because we try to find the parameters that are optimal not just for the local dataset, but also for the data residing on other devices.

Social learning studies the dynamics of learning in social networks and formulates the problem as a dynamic game between agents in a complex network. In the social network setting, people can be seen as willing participants that exchange their ideas, opinions and observations. New hypotheses emerge through our interactions and individual observations, and some of our hypotheses are invalidated with information exchange. Thus, we can easily reinterpret distributed hypothesis and social learning problem as a decentralized learning problem in the context of machine learning, since collective machine intelligence emerges through the interaction between machines. In this section, we focus on a method that is based on distributed hypothesis testing and social learning [79], and generalize this method to peer-to-peer learning [80].

Each node is able to sample local observations, or in other words have a local dataset generated by an underlying distribution $f_{\theta^*}(x)$. Peer-to-peer learning improves on the ideas first introduced in [81]. [81] proposes a novel method that only requires a strongly connected network for successful learning. It significantly loosens the assumptions on the network structure compared to previous works, where an influential nodes with uninformative observations can destabilize training. This is achieved by focusing on the neighborhood rather than the whole network. Instead of making inferences about the beliefs of other nodes in the network, nodes form their beliefs from a convex combination opinions of one-hop neighbors and their own belief. At the beginning of each communication round, nodes observe the opinions of their neighbors. Then, nodes update their Bayesian posterior belief based on their observations. Finally, they merge their belief with the opinions observed at the beginning of the communication round and form their final belief. In other words, the final belief of each node is constructed from the average belief held in the neighborhood. Lalitha [79] follow the same procedure, but the consensus average is calculated from the log-beliefs of the nodes. Nodes' influence on the consensus can be adjusted with the weights. Note that this happens prior to the learning process in the next iteration. Hence, at the next iteration learning builds upon the consensus. [82, 83] propose similar update rules with slight changes in the procedure, where [82] also analyze its consistency and characterize the convergence rate for time-varying graphs. In depth analysis of the learning rule introduced in [79] is given in [84].

Problem Setup

Until now, the problem setup was described without any mathematical notations. Here we provide the problem definition and derive a solution following [80].

Communication Network

We consider a weighted and directed graph G with a set of N individual nodes $[N]$ and a stochastic matrix W that describes the social interaction between the nodes. $N(i)$ defines the neighborhood of i and $i \in N(i)$. Moreover, if there is an edge going from j to i, then $j \in N(i)$ and i receives information from j. The influence of node j on node i is given by weight $W_{ij} \in [0,1]$ and $W_{ii} \in [0,1]$ is the confidence of i on its own knowledge. If $W_{ij} = 0$, then we can assume $j \notin N(i)$. Lastly, $\sum_{j \in N(i)} W_{ij} = 1$.

Data Distribution

Each node $i \in [N]$ has access to a local dataset D_i, where samples X_i in the local dataset are generated from the usage patterns unique to i. This means that for $i, j \in [N]$, D_i and D_j may be statistically heterogeneous. Moreover, the total number of samples in D_i is given as n_i, where $D_i = \{(X_i^0, Y_i^0), (X_i^2, Y_i^1), ..., (X_i^{n_i}, Y_i^{n_i})\}$. Number of samples at each node may not be uniform across the network, to the contrary it is more likely that data is distributed in a highly-unbalanced manner. If the data is non-identically distributed, then the local likelihood functions L_i will also be dissimilar.

Model

We assume that each user/node has the capability to learn on its local dataset, and the statistical models are structurally the same. Nodes learn the local likelihood distributions $P(D_i|\theta)$ based on their local datasets by adjusting the model's parameters $\theta \in \Theta$. Furthermore, we ensure that nodes learn a global data distribution by minimizing the objective function:

$$\arg\min_{\pi \in P(\Theta)} L_i(\pi) = E_{D_i}[P(y|x,\pi)],\ where\ P_i(y|x,\pi) = \int_{\Theta} l_i(y;x,\theta)\pi(\theta)d\theta, \tag{3.9}$$

However, calculation of this integral is intractable with complex models such as neural networks. Therefore, we approximate $P(y|x,\pi)$ by iterative optimizations on a proxy function $q(.)$, parameterized by θ. Thus, our objective function becomes:

$$\arg\min_{\pi \in P(\Theta)} E_{D_i}[KL[q(\theta)||P(y|x,\pi)]], \tag{3.10}$$

We recast this objective and make local bayesian updates to the variational posterior as follows:

$$q_i(\theta) = \arg\min_{q(.)\in\mathscr{Q}} KL[q(.)||\pi(\theta)] - E_{q(.)}[logP(x|y)] \tag{3.11}$$

The intuition behind the objective function is explained later in section 3.3.2.

Algorithm

The learning rule in peer-to-peer learning can be simplified to 3 steps as shown in algorithm 3. At each communication round nodes first make a local bayesian update and compute a new variational posterior. Secondly, nodes broadcast their posteriors to their one-hop neighbors and receive the variational posterior from the neighbors. In the final step, each node takes the weighted average of the beliefs. After the first communication round, nodes use the average belief in the neighborhood as their prior.

Intuition

The objective function must seek a degree of consensus between the nodes and robust acquisition of knowledge from the local datasets. In other we do actually seek an inexact solution for the local dataset, which actually better generalizes to the global dataset. In ordinary neural

Algorithm 3. Peer-to-Peer Learning Procedure

```
1: Input: $\pi_i^{(0)} \in P(\Theta)$ and $D_i, \forall i \in [N]$
2: for communication round $c = 1$ to C do:
3:     for $i = 1$ to N do in parallel:
4:         $q_i^{(c)}(\theta) = \arg\min_{q(.)\in\mathscr{Q}} KL[q(.)||p(y|x, \pi_i^{(c-1)})]$
5:         Broadcast $q_i^{(c)}$ to all $j \in N(i)$ and receive $q_j^{(c)}$ from all $j \in N(i)$
6:         $\pi_i^{(c)}(\theta) = \frac{exp(\sum_{j=1}^{N} W_{ij} log q_j^{(c)}(\theta))}{\int_{\Theta} exp(\sum_{j=1}^{N} W_{ij} log q_j^{(c)}(\phi)) d\phi}$
7:     end for
8: end for
```

networks, this can be achieved by regularization methods. [14, 75] provides a solution to the problem for ordinary neural networks. However, peer-to-peer learning [80] takes the bayesian approach to decentralized learning by utilizing the prior ($\pi(\theta)$) to convey the knowledge acquired by the neighbors to the nodes decisions. Each node learns from its local data and communicates its belief on the data distribution to its immediate neighbors. The first term in equation 3.11 increases the similarities between the prior and the variational posterior, and maximizing the second term maximizes the log-likelihood. Thus, during local training we seek the set of parameters that optimizes this trade-off.

After all one-hop neighbors declare their beliefs on the data distribution, the average belief of the neighborhood can be passed on to the model with the prior. Until convergence, the communication between the nodes ensures that models do not largely diverge from each other. This is forged by taking a weighted average of the log-beliefs in [84]. The prior acts as an anchor to the consensus, and limits the impact of local updates without the need to tune any parameters. Although the node does not have direct access to its neighbors datasets, we aim to pass the acquired knowledge such that it is able to identify and distinguish other prevalent data distributions than its own. It is achieved with a manner of locality, as nodes do not communicate with other parts of the network.

Implementation

In the implementation, algorithm 2 was used for the local bayesian updates in step 4. However, it is important to note that any method that accurately approximates the posterior $P(y|x,\pi)$ can be used. As explained in section 2.3, the gradient calculation of Bayes-by-Backprop method [30] is straightforward with backpropagation. More information on different methods for step 4 can be found in section 2.2.

In the consensus step (step 6), the average belief in the neighborhood is calculated. When the variational posterior $q_i(\theta)$ is a Gaussian, it is particularly easy to make the calculations. In other words, $\mathscr{Q}$ in step 2 would denote the family of Gaussian distributions. Since the average belief in the neighborhood would also be a Gaussian, we can eliminate the intractable normalization in step 6. Following [80], if we denote θ of the variational posterior q_i^c is computed as (μ_i^c, Σ_i^c), then the parameters of $\pi(\theta)$, $(\widetilde{\mu}_i^c, \widetilde{\Sigma}_i^c)$ is reduced to the following equations.

- The covariance matrices of the average belief for each node $i \in [N]$ can be calculated as:

$$\widetilde{\Sigma}_i^{(c)^{-1}} = \sum_{j \in N(i)} W_{ij} \Sigma_j^{(c)^{-1}} \tag{3.12}$$

- The mean of the average belief for each node $i \in [N]$ can be calculated as:

$$\widetilde{\mu}_i^{(c)} = \widetilde{\Sigma}_i^{(c)} \sum_{j \in N(i)} W_{ij} \Sigma_j^{(c)^{-1}} \mu_j^{(c)} \tag{3.13}$$

The effectiveness of the method is explored in the following sections.

Experiments

In this section, the performance of the peer-to-peer learning method is illustrated. Peer-to-Peer learning is one of the few methods that has shown success in fully-decentralized training of neural network models. In all of the experiments, a multi-layer perceptron architecture with 2 hidden layers of 200 neurons were used. Overall it was observed that algorithm 1 performs as good as algorithm 2. However, since algorithm 2 is much faster, KL-divergence between the variational posterior and the prior ($\pi(\theta)$) was calculated in closed-form.

Common Experiment Settings

To make thorough analysis and illustrate different aspects of peer-to-peer learning in the context of decentralized learning various experiment settings are required. Due to physical limitations we considered much smaller graphs ranging from 2 to 100 nodes. In practice, there can be millions or even billions of devices connected to the graph in various settings. However, experiments support the convergence analysis and theoretical guarantees given in [80]. In all experiments Adam optimizer with the learning rate of 0.001 was used to apply gradients to the weights at the end of each iteration.

Datasets

It can be argued that moving on to more complex datasets would be a big problem for the decentralized learning field, before establishing a robust experimental design. The lack of robust experimental design is also faced in continual learning. A recent paper criticizes the established benchmarks in the field and illustrates their weaknesses [85]. Especially for fully-decentralized learning, the lack of work also limits our abilities to construct baselines. In this work, we only demonstrate the capabilities of our method for image classification. To this end, two well-known datasets were chosen: Fashion MNIST [86] and MNIST [87].

General Effectiveness

In this section, we consider the general effectiveness of peer-to-peer learning under relaxed, but unrealistic assumptions. For this experiment, MNIST dataset was distributed independently and identically among 9 nodes in a grid topology. Each node had approximately the same number of training samples. The grid topology can be seen in figure 3.1. Weights

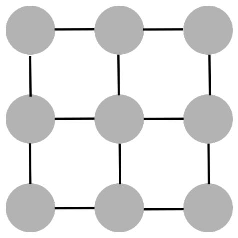


Figure 3.1. MNIST 9 Users with IID Data Experiment. MNIST dataset is independently and identically distributed over a 9-Node Graph.

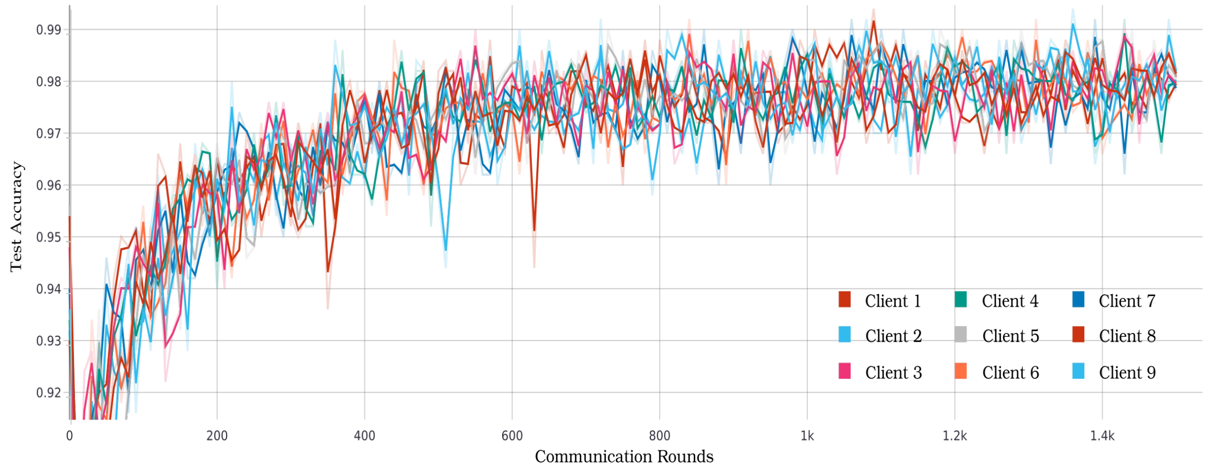


Figure 3.2. General Effectiveness with Simple Assumptions. The figure demonstrates the test accuracies of each user throughout the training. Overall, they reach 98% test accuracy, same as the baseline.

were equally distributed to the edges and according to the degrees of the nodes such that $W_{ij} = \frac{1}{|N(i)|}, \forall j \in N(i)$. For centralized approaches the baseline accuracy is 98%, and it can clearly be seen that all the nodes in the graph 3.2 are gather around the 98% test accuracy. This experiment demonstrates that peer-to-peer learning method is suitable for simple tasks with IID distribution and knowledge acquired at one node gradually disseminates to the rest of the nodes.

Effects of Non-IID and Unbalanced Data Partitioning

In this section, we consider a more difficult task than the previous experiment. In practice, datasets will reflect the usage patterns of the users. Therefore, local datasets of the nodes may be statistically heterogenous. Moreover, data may be distributed in a highly unbalanced manner. For this reason, in this experiment we explore the setting where the central node has all the

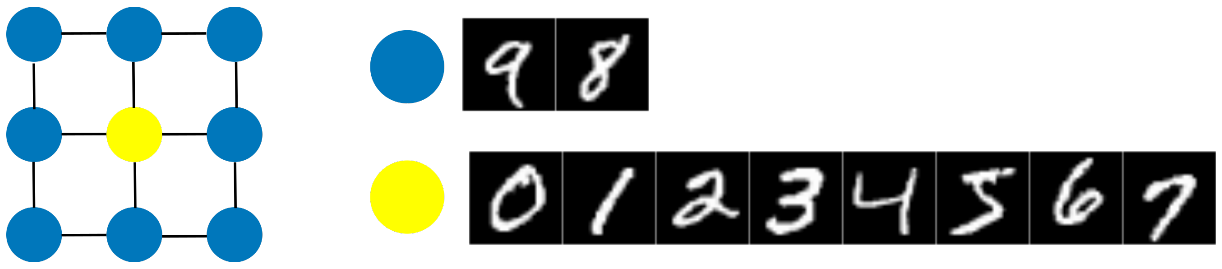


Figure 3.3. MNIST Highly-Unbalanced and Non-IID Data with Grid Topology. The yellow node has all the samples from the first 8 digits (48000 samples) and rest of the dataset is distributed equally to the blue nodes.

handwritten digits of classes 0,1,2,3,4,5,6,7 and the rest of the training dataset (classes 8,9) is randomly and equally distributed amongst the 8 remaining nodes. As a result of this non-iid distribution, there are approximately 48000 samples in the central node, while other nodes each have about 1500 samples. The topology and distribution is more clearly illustrated in figure 3.3. Social interaction matrix W remains the same as the previous experiments, all neighbors have equal influence over node $i, \forall i \in [N]$. Even under such circumstances, it can be seen from figure 3.4 that there is little change in the learning curve and all the nodes are gathered aroun 98% test accuracy. Unfortunately, peer-to-peer learning method is not successful in all cases. We will see in the following experiments that the success of the experiment can be attributed to the careful placement of the classes.

Effects of User Placement and Influence

The previous experiment demonstrated the performance of peer-to-peer learning method on a more complicated task, where data was distributed highly non-iid and unbalanced. We also assumed that the most-informative node would also be placed at the most influential place in the graph. In the grid-like structure seen in figure 3.5, the central node is the most influential node when all nodes are affected equally by their immediate neighbors. Here we investigate a different situation, where the most informative node is at the least influential spot at the grid. In this experiment, only the place of the most informative node is changed. Other variables remain the same. The results are demonstrated in figure 3.6. When the users are placed according to

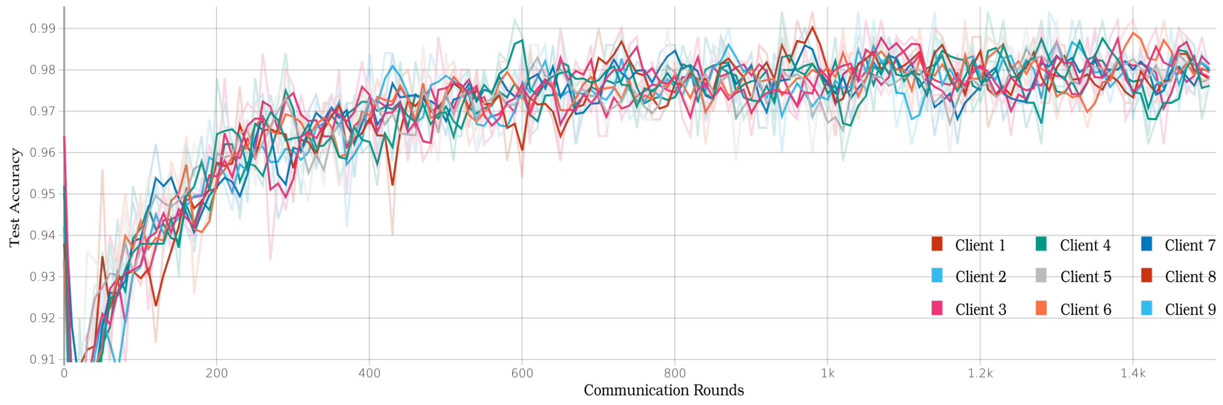


Figure 3.4. MNIST 9 Users with Unbalanced and Non-IID Data Experiment. A grid-like graph structure was simulated where each row and column had 3 users connected to each other. The central node had 48000 samples from classes 0,1,2,3,4,5,6,7 and the remaining samples were distributed evenly amongst the rest of the nodes. All the nodes reach 98% overall accuracy, same as the baseline.

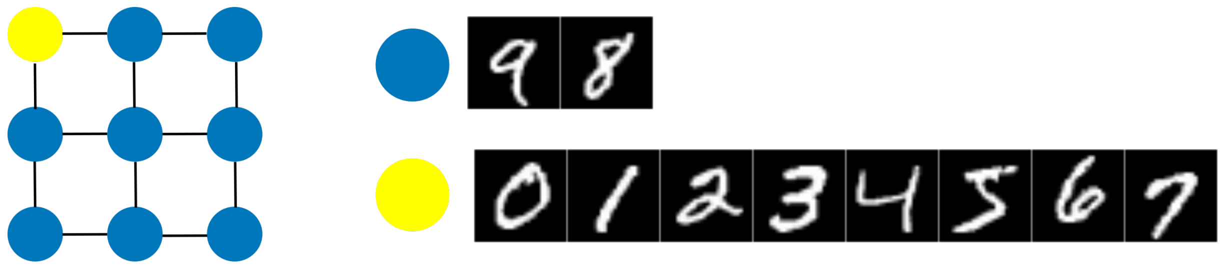


Figure 3.5. Most Informative Node at the Least Influential Spot Experiment Setting. The yellow node has 48000 samples from classes 0,1,2,3,4,5,6,7 and the remaining samples are distributed evenly to the blue nodes. Due to the social interaction matrix, in this case corner nodes are the least influential nodes in this grid-like structure, where as center nodes are the most influential.

their local datasets informativeness, the convergence rate would be higher. For example, in the previous section, we went through a case, where the most informative node was placed at the center. Given the social interaction matrix, where edges on each row are equally weighted, the central node is the most influential node over all the network. Thus, we can expect higher convergence rates. Due to the simplicity of the task, the difference between figure 3.3 and figure 3.5 is not noticable. The learning curve is similar and final test accuracy of the nodes (98%) are negligibly close. The social interaction matrix and node placement are both effective for the resulting influences of the nodes. In the next section, using a different topology we analyze the effects of social interaction matrix.

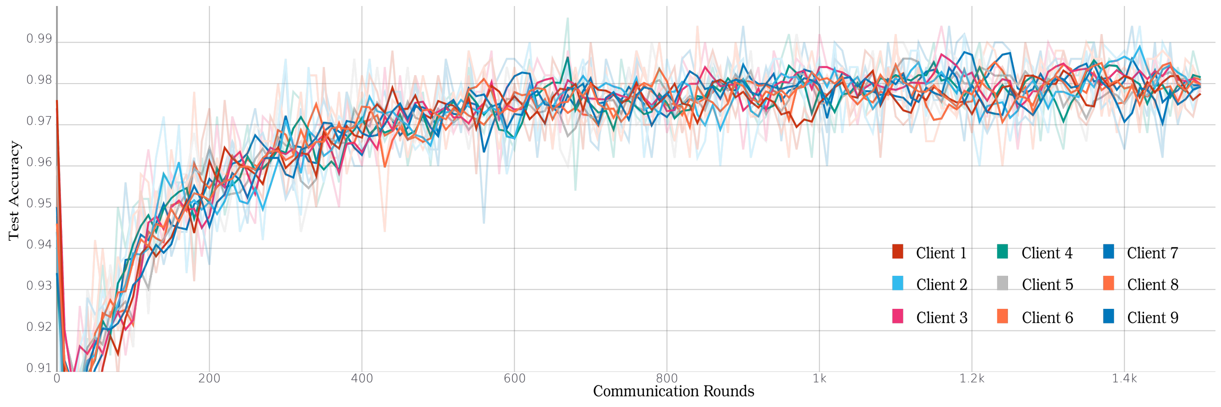


Figure 3.6. Most Informative Node at the Least Influential Spot. All the nodes reach 98% overall accuracy. Information flows regardless where the most informative node is located in the graph. If a node learns to distinguish between the classes, no matter where it is placed on the graph, the acquired knowledge will spread through the network.

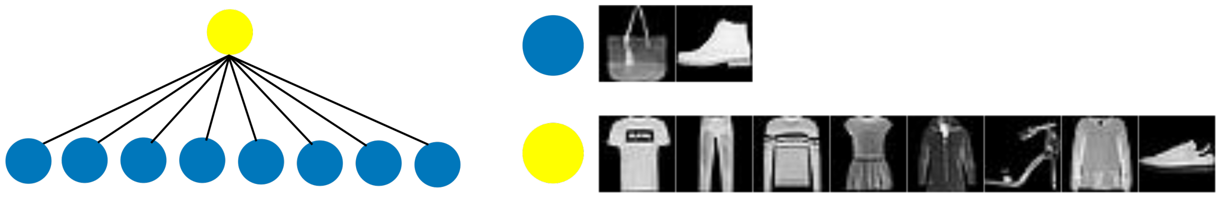


Figure 3.7. Star Topology and Fashion MNIST Data Distribution for Weight Matrix Analysis. The most informative node is the yellow node with 48000 samples from 8 classes, whereas blue nodes have 1500 samples from 2 classes. The confidence factor of the blue nodes is given by α and blue node's confidence

Effects of the Weight Matrix

In this section, we investigate how the social interaction matrix affects learning. It is important to note that the influence of the nodes in the network depends on the eigenvector centrality of the social interaction matrix. The eigenvector centrality of a stochastic matrix such as W can be easily calculated by taking the power of W until convergence. The eigenvector centrality should be fair to give each node to have an influence over the global learning. It should not vanish nor should it explode. This was an important issue to address before running any tests. Especially for larger graphs, it becomes harder to assign individual weights to each edge. For this reason, the experiment setting with only 9 nodes was created as shown in figure 3.7. In this experiment we use Fashion MNIST dataset, where the training dataset includes 60000

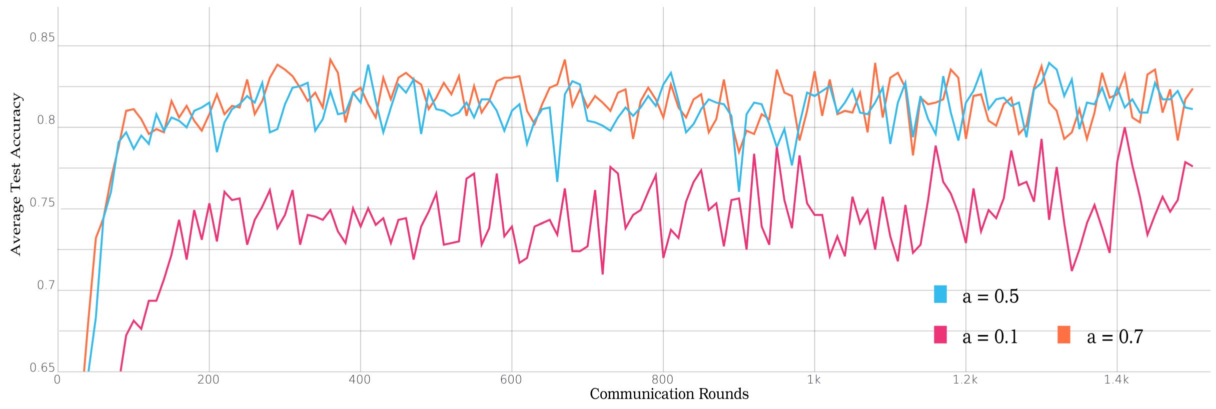


Figure 3.8. Effects of the Weight Matrix. Social interaction matrix reflects the confidence on the beliefs. When badly designed, the performance of the method suffers.

samples from 10 different clothing classes. The number of samples from the first 8 classes are 48000 and the last two classes have 12000 samples in total. The first node colored with yellow contains all the samples from the first 8 classes, and blue nodes share the rest of the training dataset randomly and equally. Yellow node can be seen as the master node, since it has a similar role as the central server in federated learning. It is connected to all the other nodes in the graph, and information from one node to another can only flow through it. The row corresponding to the yellow node in the social interaction matrix is given as $W_1 = [1/9, ..., 1/9]$, where all the nodes have equal influence on the average belief. For the blue nodes weights are assigned as; $W_{i1} = \alpha$ and $W_{ii} = 1 - \alpha$. Since blue nodes are only connected to the yellow node, all other elements on their corresponding row in W would take zero. Compared to MNIST, Fashion MNIST includes more complex texture and shapes. Therefore, with the MLP architecture achieves 88% accuracy with centralized training. With this setting, we expect the convergence rate to grow, as α increases. Indeed, it can be clearly seen from figure 3.8 that this is the case. When $\alpha = 0.1$, the average test accuracy converges to 77.5%, which is nearly 10% lower than the baseline. It can be seen that the more confidence we place on the most informative node, the closer the learning curve converges to the baseline. Thus, the design of the social interaction matrix highly affects the performance of learning. Unfortunately, in very large graphs it becomes especially harder to design such a graph. Moreover, the decentralized learning graph would be dynamic in practice and data distributions will change quite often. The social interaction matrix, and the ability to assign different confidence levels to neighbors, enables a framework for *explore and exploit* in learning. For example it can be leveraged to adjust dissimilarity between the models and make them more personalized. It can also be seen that the average test accuracy over all the test dataset is affected highly affected by such adjustment. Therefore, the design of the weight matrix requires a lot of thought and it is an interesting problem that should be robustly addressed by future work.

Effects of Dissimilar Data Distributions

Decentralized learning will face various challenging data distributions in the real world. In this section, we investigate if peer-to-peer learning is robust against different data partitions. To this end, we use Fashion MNIST dataset which is comparatively more complex than MNIST. In this case, we do not want a complex graph topology nor do we want an unbalanced distribution. Therefore, we create a 10-node directed circular graph, where two types of nodes placed in an alternating fashion. There are 5 nodes of type-1 and 5 nodes of type-2. Odd nodes are type 1 and even nodes are type-2. Nodes only receive information from their clockwise neighbor and only send information to their counter-clockwise neighbor. Social interaction matrix was adjusted such that nodes have more confidence in their neighbor than themselves. In particular, $W_{ii} = 0.25$ and $W_{ij} = 0.75$, where $j = i+1 \; \forall \; i < 10$ and $j = 1$ for node 10. The rest of the elements in W are zero. Since we aim to test how peer-to-peer learning performs with different data partitions, we made two experiments with different data partitioning strategies. Even though the data partitioning was changed very slightly, the performance strangely suffered under the second case. In the first case, we consider a situation where all the resembling classes are collected under the same type of node. The graph structure and class selection can be seen on figure 3.9. All samples from the *shirt-like* classes are independently and identically distributed to yellow nodes (type-1). The remaining samples from the classes including three *shoe-like* classes, trouser and bag are independently and identicaly distributed to blue nodes (type-2).

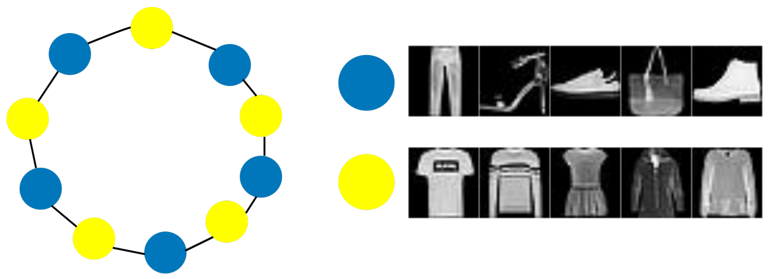


Figure 3.9. Dissimilar Data Distribution and Circular Graph Topology. Samples from all resembling classes are collected under one type.

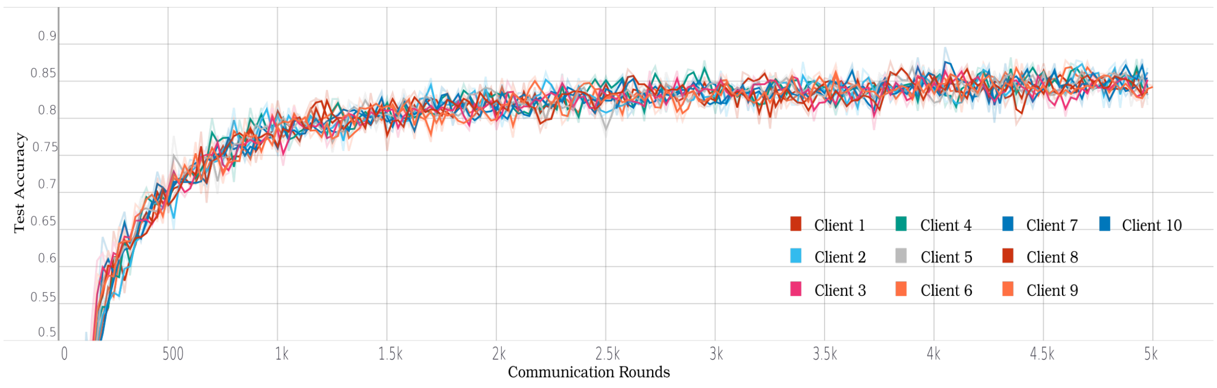


Figure 3.10. Effects of Very Dissimilar Data Distributions. All nodes achieve a test accuracy around 85%.

Results are demonstrated in figure 3.10. It can be seen that even with the sparsity of the graph average test accuracy converges to 85%. There is a slight decrease in test accuracy ($\sim$3%) compared to the centralized learning method. This can still be considered a successful result given that there was little optimization made for the learning. In the first case, we consider a situation where all the resembling classes are collected under the same type of node. The graph structure and class selection can be seen on figure 3.9. All samples from the *shirt-like* classes are independently and identically distributed to yellow nodes (type-1). The remaining samples from the classes including three *shoe-like* classes, *trouser* and *bag* are independently and identicaly distributed to blue nodes (type-2). It can be seen that even with the sparsity of the graph average test accuracy converges to 85%. There is a slight decrease in test accuracy ($\tilde{3}$%) compared to the centralized learning method. This can still be considered a successful result given that there was little optimization made for the learning. Although the results of the first

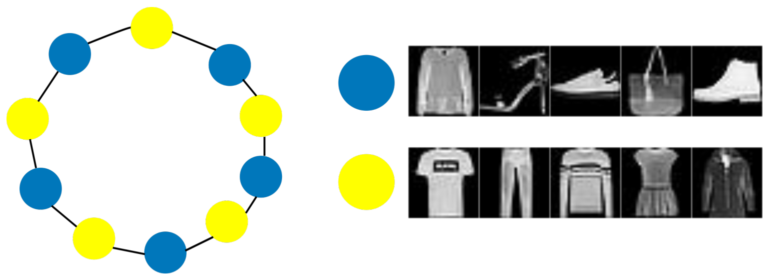


Figure 3.11. Dissimilar Data Distribution and Circular Graph Topology. *shirt* and *trouser* classes are exchanged between types. Blue nodes also learn the distribution of a shirt-like class.

case may raise the expectations from the model, peer-to-peer learning underperforms for the second case. In the first case, all resembling classes were represented on the same nodes. The other nodes did not have any samples from these classes. We now consider a situation where this is not case. The graph structure and class selection can be seen on figure 3.11. All samples from the *shirt-like* classes except the *shirt* class and sampels from the trouser class are independently and identically distributed to yellow nodes (type-1). The remaining samples from the classes including three *shoe-like* classes, *shirt* and *bag* are independently and identicaly distributed to blue nodes (type-2). Notice that only the *shirt* class and the *trouser* class are exchanged between the two types. The number of samples at each node is equal. Moreover, now blue nodes have samples from a *shirt-like* class in their local dataset.

The results for the experiment are demonstrated in figure 3.12. While there is only a negligible decrease in accuracy for the yellow nodes, the blue nodes (even nodes) strangely do not seem to be learning. The average test accuracy of the blue nodes is 67% about 15% lower than yellow nodes. Nodes ability to distinguish the classes and classify them correctly are demonstrated in figure 3.13. It can be clearly seen from the confusion matrices in figure 3.13 that nodes are highly biased towards their own datasets. Nodes are not accurately learning to distinguish between the classes, when they have samples from resembling classes in their own dataset. This is because peer-to-peer learning is suffering from a problem also common in prior-based continual learning approaches [85]. In continual learning prior-based methods are successful in transferring learning between the tasks, if the tasks are dissimilar and do not

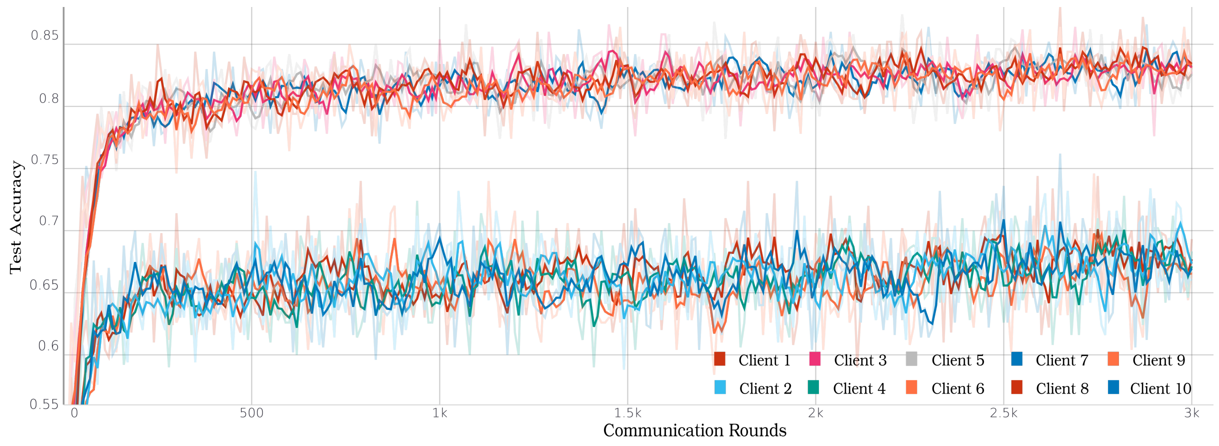
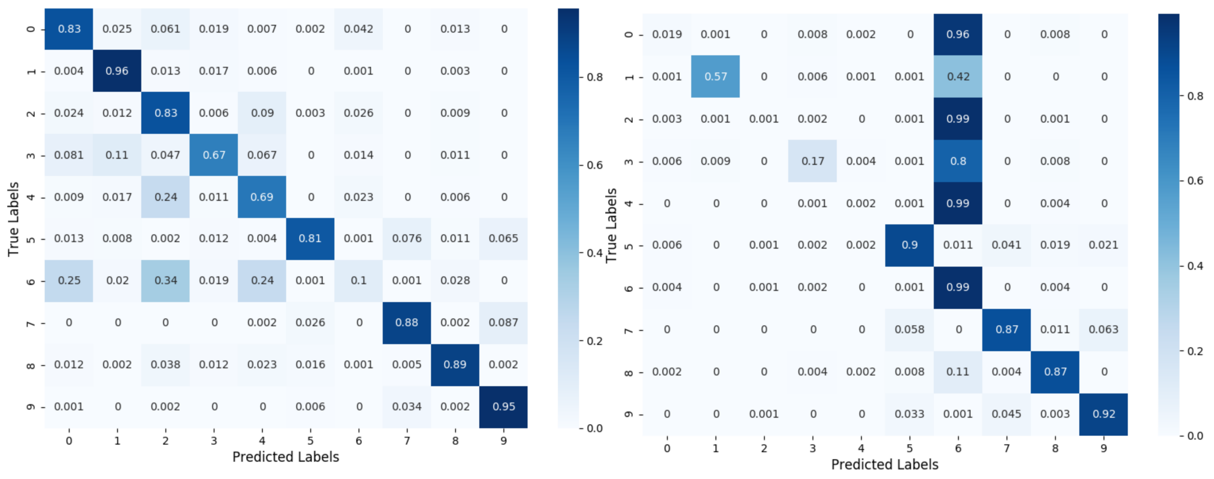


Figure 3.12. Fashion MNIST Non-IID Data Experiment with Resembling Classes.

resemble each other. However, they fail when it's not the case. Similarly, peer-to-peer learning method is successful, when all the samples from the resembling classes are in the same dataset. Unfortunately, in other cases, it fails. It's critical that the prior of the local bayesian model adequately represents the uncertainty and conveys this uncertainty to the decisions made by the model. For the good case, peer-to-peer learning is successful because of the dissimilarity in data distributions. In bayesian learning, we expect the decisions to reflect the uncertainty of the prediction. Since learning builds on the prior, especially at the beginning of each communication round, model's decision would depend heavily on the prior. In the good case, since datasets are very dissimilar, model would make inconfident decisions and the gradients would be small. Due to this regulatory effect from the prior, the model would learn gradually. However, when there are resembling classes, at the beginning of the training models would confident incorrect decisions. When a model makes a confident prediction, the gradients would be proportionally larger. After the first communication round, all nodes would have knowledge about the average belief in their neighborhood and this would be assigned as their prior for the next communication round. The first decision models make would be heavily biased towards the average belief of the neighborhood. Normally, we wish the gradients to change the weights such that the model becomes more successful in classifying the samples. Let's illustrate how the gradients would affect learning with an example. After the first communication round the blue node would



(a) Type-1 Node Confusion Matrix

(b) Type-2 Node Confusion Matrix

Figure 3.13. Confusion Matrices of Two Nodes with Resembling Class Bias. The confusion matrix on the left side demonstrates that type-1 nodes do not learn the encoding of the samples from ”shirt” class (label *6*) and is heavily biased towards the shirt-like classes that are represented in its local dataset. Confusion matrix on the right side shows that type-2 nodes become heavily biased towards ”shirt” class (label *6*)and incorrectly classify the shirt-like classes as ”shirt”.

have some information about the shirt-like classes in the yellow nodes. During local learning, it would get a sample from the *shirt* class, which heavily resembles the shirt-like samples in the yellow node. It would make a confident but incorrect decision on the *shirt* sample. Thus, the gradients that are essentially used to classify *shirt* would be large, but also weights that distinguish other *shirt-like* classes would be heavily affected. If the uncertainty was well-reflected to the decisions, this would not have been a problem, learning would be stable and successful. Unfortunately, peer-to-peer learning is unsuccessfuly for these cases due to the accumulation of gradient mis-estimates. This shows a common challenge faced by both prior-based continual learning aproaches and bayesian decentralized learning. The gradient mis-calculations are the main culprit for the poor performance of peer-to-peer learning. The idea implemented in peer-to-peer learning is quite intuitive, where we use the average belief of the neighborhood as the prior to learning in the next communication round. In the context of bayesian decentralized learning, approximation of the local variational posteriors affect the global learning as well. Thus, a more accurate approximation should be provided in order to make peer-to-peer learning more practical.

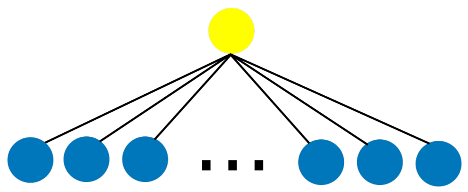


Figure 3.14. Bayesian Decentralized Asynchronous Learning Graph Topology. Yellow node acts as the central server; it does not have a local dataset and it is used to construct the global model. A subset of the blue nodes are selected at each communication round to be trained.

This is an interesting challenge that may be addressed by future work.

Asynchronous Training

As the communication network gets larger, it becomes impossible to coordinate the users to train simultaneously. Therefore, decentralized learning methods that enable asynchronous learning is more favorable. In this section, we analyze the performance of bayesian decentralized asynchronous learning based on peer-to-peer learning and federated learning. Graph structure can be seen in figure 3.14. Only a subset of the network train their models simultaneously. Yellow node acts as the central server in federated learning; it is only used to accumulate knowledge and transfer it between the nodes. All nodes have equal influence over the global model, i.e. $W_{1i} = 1/100 \forall i \in [N], i \neq 1$. Moreover, the training is initialized with the global model such that $W_{i1} = 1, \forall i \in [N], i \neq 1$ and all other elements in the weight matrix is zero. Similar to federated learning, at each communication round a subset of the nodes are randomly selected and trained in parallel. We use a 100-node graph with a star topology. MNIST dataset was distributed equally to the blue nodes, but to test the robustness of the method several different data distributions was considered. Each client trains for only an epoch with a batch size of 10 except the third case, where models were trained with a batch size of 128. Results can be seen from figure 3.15. In the figure, line plots demonstrate the average test accuracy of the blue nodes and test accuracy of the yellow node for different cases. When data is distributed in an iid manner among the blue nodes, the method was largely successful. We first experimented with the case, where only half of the clients were trained at the same time. In the second case, 30% of the clients were selected at each

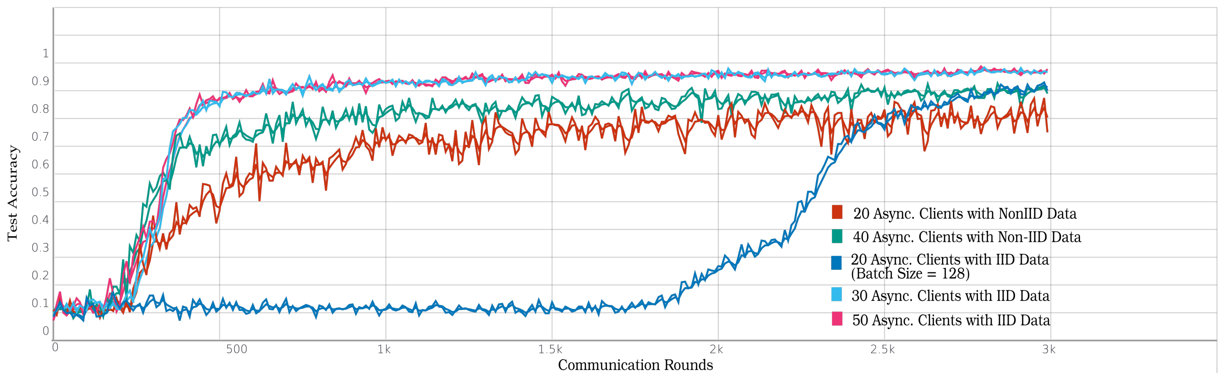


Figure 3.15. Bayesian Decentralized Asynchronous Learning.

communication round. It can be seen that in both cases test accuracies of the models reach the baseline test accuracy (98%). In the third case, once again we randomly distribute 600 samples to each blue node, but only 20 nodes are trained at the same time and a batch size of 128 is used. It can be seen that the model starts converging much later in training. Given that the learning curve did not change in the first two cases, this can be due to the scarcity of local updates. In the fourth case, the training dataset was sorted by label and split into 500 shards of size 120. After the shards were shuffled, they were distributed to the blue nodes. Each blue node received 5 shards with 600 samples in total. Although we increased the number asynchronous clients to 40, the method performs poorly with only 80% test accuracy. This is due to the inherent issues of peer-to-peer learning. When dissimilarity between datasets are small, nodes fail to distinguish the similar data distributions that is not represented in their own local dataset. In the fifth case, the training dataset was sorted by label and split into 200 shards of size 300. After the shards were shuffled, they were distributed to the blue nodes. Each blue node received 2 shards with 600 samples in total. At each communication round, 20 nodes were selected and trained in parallel. The performance is much worse than all other cases. The test accuracy converged to 70% test accuracy.

The method used in this experiment is only a special case of peer-to-peer learning. Although it performed poorly for some of the cases, this is largely due to problems explained in section 3.4.6. If this problem can be addressed, our experiments suggest that peer-to-peer learning is amenable to asynchronicity.

Chapter 4

Conclusion

Decentralized learning is a growing field that requires more attention from the research community. It is not just important for the democratization of machine intelligence, but it is also a less expensive method to train powerful statistical models. Due to its setting and scale, there are many interesting challenges to be solved.

In this work, we approached decentralized learning from a bayesian learning perspective, which enables the embodiment of all available knowledge in the model. It provides an automatic Occam's Razor, i.e. it does not require regularization methods as Maximum-Likelihood Estimation does. It inherently has the ability to convey the uncertainties to the decisions that the model makes. For this reason, it can be used to make better calibrated and more informed decisions. This especially important for deep learning models, where interpretability of the model is limited and individual roles of model parameters are unknown. There are many methods to approach deep learning from a bayesian perspective. In section 2.3, we attempted to explain some of the methods.

To the best of our knowledge, peer-to-peer learning is the first bayesian approach to decentralized learning [80]. Moreover, some existing prior-based approaches in continual learning can be generalized to it [88, 89]. It has been demonstrated by the experiments in the previous chapter that peer-to-peer learning is generally effective, but it can be further improved.

Future Work

Decentralized learning is open to improvements from many aspects. Complex challenges faced in the field require creative solutions. These challenges include secure multi party computation of the global updates, differential privacy, the optimization of the social interaction matrix and improvements for posterior approximation in bayesian parameter estimation.

Secure Computation and Differential Privacy

In general, it is best to restrict direct access to user models. Moreover, communication between neighbors must be secure. This is only possible through encryption. In federated learning, there are some practical methods to achieve secure aggregation of gradient updates [90]. However, it remains an important problem in decentralized learning. Methods that target this problem should make ensure secure knowledge sharing and secure computations for model updates.

Decentralized learning shrinks the risk surface for data-leaks as users won't be sharing their knowledge with a central entity. However, it is possible to infer the samples back from the statistical models. This is because models learn the encoding of local data distributions. In other words, they carry an encoding of the samples they were trained on. For this reason, differential privacy has become a trending topic in machine learning. In the context of decentralized learning, knowledge learnt at one node spreads through the graph and all models would eventually embody the knowledge learnt at other nodes. Therefore, addressing this problem is of utmost importance to protect user's privacy.

Social Interaction Matrix

In federated averaging, the heuristic for weights is incredibly simple. Given its simplicity and the lack of theoretical guarantees, it still demonstrates very successful results. This is not the case in peer-to-peer learning. Optimal weight matrix depends on the graph structure, data distribution and the complexity of the tasks at hand. In practice, the social interaction matrix would also be dynamic. It is unrealistic to assume that is given. The problem is more challenging where graph is large and nodes only have access to information about their one-hop neighbors.

Gradient Estimation

In the previous chapter, it was demonstrated that peer-to-peer learning method performs poorly for non-iid data. This is because the uncertainty is not well-represented by the prior and it is not conveyed into the model's decisions. As a result, the gradients are misestimated and overall test accuracy of the models suffer. Thus, to address this problem gradients must be more accurately estimated with a better representation of the uncertainty. To this end, methods that more closely approximate the posterior can increase the performance of peer-to-peer learning in the local bayesian update step. One such method is Bayesian Gradient Descent, which was proposed for task agnostic continual learning [91].

Conclusion

In this work, we introduced decentralized learning, bayesian neural networks, and implemented a bayesian decentralized learning method. Peer-to-peer learning provides the theoretical guarantees that federated learning does not. However, it must be improved to be practical. To the best of our knowledge, there is very little work in fully-decentralized learning that targets neural networks and complex tasks. Introduction of new methods would stimulate innovation and decentralized learning can become a driving force behind machine intelligence. We are at the beginning of a special phase for machine learning.

Bibliography

[1] S. Luo, H. Xia, T. Yoshida, and Z. Wang, “Toward collective intelligence of online communities: A primitive conceptual model,” *Journal of Systems Science and Systems Engineering*, vol. 18, no. 2, pp. 203–221, 2009.

[2] B. Kogut and A. Metiu, “Open-source software development and distributed innovation,” *Oxford review of economic policy*, vol. 17, no. 2, pp. 248–264, 2001.

[3] D. H. Wolpert and K. Tumer, “An introduction to collective intelligence,” *arXiv preprint cs/9908014*, 1999.

[4] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[5] D. P. Kingma, *Variational inference & deep learning: A new synthesis*. PhD thesis, PhD thesis, University of Amsterdam, 2017.

[6] A. K. Jain and B. Chandrasekaran, “39 dimensionality and sample size considerations in pattern recognition practice,” *Handbook of statistics*, vol. 2, pp. 835–855, 1982.

[7] S. J. Raudys and A. K. Jain, “Small sample size effects in statistical pattern recognition: Recommendations for practitioners,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 3, pp. 252–264, 1991.

[8] A. Halevy, P. Norvig, and F. Pereira, “The unreasonable effectiveness of data,” *IEEE Intelligent Systems*, 2009.

[9] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, “Revisiting unreasonable effectiveness of data in deep learning era,” in *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017.

[10] D. Steinkraus, I. Buck, and P. Simard, “Using gpus for machine learning algorithms,” in *Eighth International Conference on Document Analysis and Recognition (ICDAR’05)*, pp. 1115–1120, IEEE, 2005.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[12] A. Acquisti, A. Friedman, and R. Telang, “Is there a cost to privacy breaches? an event study,” *ICIS 2006 Proceedings*, p. 94, 2006.

[13] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, “Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent,” in *Advances in Neural Information Processing Systems*, pp. 5330–5340, 2017.

[14] P. Vanhaesebrouck, A. Bellet, and M. Tommasi, “Decentralized collaborative learning of personalized models over networks,” in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[15] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” 2016.

[16] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.

[17] D. J. MacKay, *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

[18] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.

[19] Y. Gal, *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge, 2016.

[20] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[21] Z. Ghahramani, “Probabilistic machine learning and artificial intelligence,” *Nature*, vol. 521, no. 7553, p. 452, 2015.

[22] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1321–1330, JMLR. org, 2017.

[23] C. E. Rasmussen and Z. Ghahramani, “Occam’s razor,” in *Advances in neural information processing systems*, pp. 294–300, 2001.

[24] C. M. Bishop, “Regularization and complexity control in feed-forward networks,” in *Proceedings International Conference on Artificial Neural Networks ICANN*, vol. 95, pp. 141–148, 1995.

[25] J. E. Moody, “The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems,” in *Advances in neural information processing systems*, pp. 847–854, 1992.

[26] E. Byvatov, U. Fechner, J. Sadowski, and G. Schneider, “Comparison of support vector machine and artificial neural network systems for drug/nondrug classification,” *Journal of chemical information and computer sciences*, vol. 43, no. 6, pp. 1882–1889, 2003.

[27] I. Yilmaz, “Comparison of landslide susceptibility mapping methodologies for koyulhisar, turkey: conditional probability, logistic regression, artificial neural networks, and support vector machine,” *Environmental Earth Sciences*, vol. 61, no. 4, pp. 821–836, 2010.

[28] I. J. Myung, “The importance of complexity in model selection,” *Journal of mathematical psychology*, vol. 44, no. 1, pp. 190–204, 2000.

[29] Y. Gal and Z. Ghahramani, “Bayesian convolutional neural networks with bernoulli approximate variational inference,” *arXiv preprint arXiv:1506.02158*, 2015.

[30] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, “Weight uncertainty in neural networks,” *arXiv preprint arXiv:1505.05424*, 2015.

[31] N. Metropolis and S. Ulam, “The monte carlo method,” *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[32] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[33] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, “A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains,” *The annals of mathematical statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[34] A. E. Gelfand and A. F. Smith, “Sampling-based approaches to calculating marginal densities,” *Journal of the American statistical association*, vol. 85, no. 410, pp. 398–409, 1990.

[35] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, “An introduction to mcmc for machine learning,” *Machine learning*, vol. 50, no. 1-2, pp. 5–43, 2003.

[36] G. Casella and E. I. George, “Explaining the gibbs sampler,” *The American Statistician*, vol. 46, no. 3, pp. 167–174, 1992.

[37] J. Gill, “Is partial-dimension convergence a problem for inferences from mcmc algorithms?,” *Political Analysis*, vol. 12, no. 4, 2004. PDF.

[38] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, “Variational inference: A review for statisticians,” *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[39] D. M. Blei and M. I. Jordan, “Variational inference for dirichlet process mixtures,” *Bayesian analysis*, vol. 1, no. 1, pp. 121–143, 2006.

[40] A. Graves, "Practical variational inference for neural networks," in *Advances in neural information processing systems*, pp. 2348–2356, 2011.

[41] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

[42] J. N. Tsitsiklis, *Problems in decentralized decision making and computation.* PhD thesis, Massachusetts Institute of Technology, 1984.

[43] J. Tsitsiklis, D. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE transactions on automatic control*, vol. 31, no. 9, pp. 803–812, 1986.

[44] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[45] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pp. 482–491, IEEE, 2003.

[46] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: Design, analysis and applications," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 3, pp. 1653–1664, IEEE, 2005.

[47] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2508–2530, 2006.

[48] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, p. 48, 2009.

[49] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Transactions on Automatic control*, vol. 57, no. 3, pp. 592–606, 2011.

[50] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, pp. 2595–2603, 2010.

[51] S. S. Ram, A. Nedić, and V. V. Veeravalli, "Asynchronous gossip algorithms for stochastic optimization," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pp. 3581–3586, IEEE, 2009.

[52] S. S. Ram, A. Nedić, and V. V. Veeravalli, "Asynchronous gossip algorithm for stochastic optimization: Constant stepsize analysis," in *Recent Advances in Optimization and its Applications in Engineering*, pp. 51–60, Springer, 2010.

[53] K. Srivastava and A. Nedic, “Distributed asynchronous constrained stochastic optimization,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 4, pp. 772–790, 2011.

[54] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Ng, “Large scale distributed deep networks,” in *Advances in neural information processing systems*, pp. 1223–1231, 2012.

[55] S. Zhang, A. E. Choromanska, and Y. LeCun, “Deep learning with elastic averaging sgd,” in *Advances in Neural Information Processing Systems*, pp. 685–693, 2015.

[56] H.-P. Cheng, P. Yu, H. Hu, F. Yan, S. Li, H. Li, and Y. Chen, “Leasgd: an efficient and privacy-preserving decentralized algorithm for distributed learning,” *arXiv preprint arXiv:1811.11124*, 2018.

[57] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, “D2: Decentralized training over decentralized data,” *arXiv preprint arXiv:1803.07068*, 2018.

[58] K. Yuan, Q. Ling, and W. Yin, “On the convergence of decentralized gradient descent,” *SIAM Journal on Optimization*, vol. 26, no. 3, pp. 1835–1854, 2016.

[59] W. Shi, Q. Ling, G. Wu, and W. Yin, “Extra: An exact first-order algorithm for decentralized consensus optimization,” *SIAM Journal on Optimization*, vol. 25, no. 2, pp. 944–966, 2015.

[60] A. Mokhtari and A. Ribeiro, “Dsa: Decentralized double stochastic averaging gradient algorithm,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2165–2199, 2016.

[61] M. Schmidt, N. Le Roux, and F. Bach, “Minimizing finite sums with the stochastic average gradient,” *Mathematical Programming*, vol. 162, no. 1-2, pp. 83–112, 2017.

[62] A. Defazio, F. Bach, and S. Lacoste-Julien, “Saga: A fast incremental gradient method with support for non-strongly convex composite objectives,” in *Advances in neural information processing systems*, pp. 1646–1654, 2014.

[63] L. He, A. Bian, and M. Jaggi, “Cola: Decentralized linear learning,” in *Advances in Neural Information Processing Systems*, pp. 4536–4546, 2018.

[64] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan, “Communication-efficient distributed dual coordinate ascent,” in *Advances in neural information processing systems*, pp. 3068–3076, 2014.

[65] V. Cevher, S. Becker, and M. Schmidt, “Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics,” *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 32–43, 2014.

[66] A. T. Suresh, F. X. Yu, S. Kumar, and H. B. McMahan, “Distributed mean estimation with limited communication,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3329–3337, JMLR. org, 2017.

[67] O. Shamir, N. Srebro, and T. Zhang, “Communication-efficient distributed optimization using an approximate newton-type method,” in *International conference on machine learning*, pp. 1000–1008, 2014.

[68] R. Johnson and T. Zhang, “Accelerating stochastic gradient descent using predictive variance reduction,” in *Advances in neural information processing systems*, pp. 315–323, 2013.

[69] J. Konečnỳ, “Stochastic, distributed and federated optimization for machine learning,” *arXiv preprint arXiv:1707.01155*, 2017.

[70] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[71] E. Wei and A. Ozdaglar, “Distributed alternating direction method of multipliers,” in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 5445–5450, IEEE, 2012.

[72] A. Makhdoumi and A. Ozdaglar, “Broadcast-based distributed alternating direction method of multipliers,” in *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 270–277, IEEE, 2014.

[73] R. Zhang and J. Kwok, “Asynchronous distributed admm for consensus optimization,” in *International Conference on Machine Learning*, pp. 1701–1709, 2014.

[74] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, “Learning with local and global consistency,” in *Advances in neural information processing systems*, pp. 321–328, 2004.

[75] A. Bellet, R. Guerraoui, M. Taziki, and M. Tommasi, “Personalized and private peer-to-peer machine learning,” *arXiv preprint arXiv:1705.08435*, 2017.

[76] V. Zantedeschi, A. Bellet, and M. Tommasi, “Communication-efficient and decentralized multi-task boosting while learning the collaboration graph,” *arXiv preprint arXiv:1901.08460*, 2019.

[77] C. Shen and H. Li, “On the dual formulation of boosting algorithms,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 12, pp. 2216–2231, 2010.

[78] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. Van Steen, “Gossip-based peer sampling,” *ACM Transactions on Computer Systems (TOCS)*, vol. 25, no. 3, p. 8, 2007.

[79] A. Lalitha, A. Sarwate, and T. Javidi, “Social learning and distributed hypothesis testing,” in *2014 IEEE International Symposium on Information Theory*, pp. 551–555, IEEE, 2014.

[80] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar, “Peer-to-peer federated learning on graphs,” 2019.

[81] A. Jadbabaie, P. Molavi, A. Sandroni, and A. Tahbaz-Salehi, "Non-bayesian social learning," *Games and Economic Behavior*, vol. 76, no. 1, pp. 210 – 225, 2012.

[82] A. Nedić, A. Olshevsky, and C. A. Uribe, "Nonasymptotic convergence rates for cooperative learning over time-varying directed graphs," in *2015 American Control Conference (ACC)*, pp. 5884–5889, IEEE, 2015.

[83] S. Shahrampour, A. Rakhlin, and A. Jadbabaie, "Distributed detection: Finite-time analysis and impact of network topology," *IEEE Transactions on Automatic Control*, vol. 61, no. 11, pp. 3256–3268, 2015.

[84] A. Lalitha, T. Javidi, and A. D. Sarwate, "Social learning and distributed hypothesis testing," *IEEE Transactions on Information Theory*, vol. 64, no. 9, pp. 6161–6179, 2018.

[85] S. Farquhar and Y. Gal, "Towards robust evaluations of continual learning," *arXiv preprint arXiv:1805.09733*, 2018.

[86] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[87] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, p. 18, 2010.

[88] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner, "Variational continual learning," *arXiv preprint arXiv:1710.10628*, 2017.

[89] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3987–3995, JMLR. org, 2017.

[90] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, ACM, 2017.

[91] C. Zeno, I. Golan, E. Hoffer, and D. Soudry, "Task agnostic continual learning using online variational bayes," *arXiv preprint arXiv:1803.10123*, 2018.