
Open Access

*Open Journal of Internet of Things (OJIOT)*
*Volume 3, Issue 1, 2017*

http://www.ronpub.com/ojiot
**ISSN 2364-7108**

# Differentially Private Linear Models for Gossip Learning through Data Perturbation

István Hegedűs, Márk Jelasity

University of Szeged and MTA-SZTE Research Group on AI, Dugonics square 13, H-6720 Szeged, Hungary,
{ihegedus,jelasity}@inf.u-szeged.hu

## Abstract

*Privacy is a key concern in many distributed systems that are rich in personal data such as networks of smart meters or smartphones. Decentralizing the processing of personal data in such systems is a promising first step towards achieving privacy through avoiding the collection of data altogether. However, decentralization in itself is not enough: Additional guarantees such as differential privacy are highly desirable. Here, we focus on stochastic gradient descent (SGD), a popular approach to implement distributed learning. Our goal is to design differentially private variants of SGD to be applied in gossip learning, a decentralized learning framework. Known approaches that are suitable for our scenario focus on protecting the gradient that is being computed in each iteration of SGD. This has the drawback that each data point can be accessed only a small number of times. We propose a solution in which we effectively publish the entire database in a differentially private way so that linear learners could be run that are allowed to access any (perturbed) data point any number of times. This flexibility is very useful when using the method in combination with distributed learning environments. We show empirically that the performance of the obtained model is comparable to that of previous gradient-based approaches and it is even superior in certain scenarios.*

## Type of Paper and Keywords

Regular research paper: *distributed differential privacy, stochastic gradient descent, linear models, machine learning, distributed learning, gossip learning*

## 1 Introduction

We are witnessing the proliferation of distributed computer systems that are composed of devices rich in sensitive personal data. These include the Internet of Things, and networks of smart meters and smartphones. While mining personal data has very important applications, data mining algorithms must protect the privacy of the users involved. An emerging approach to process data in such systems is relying on decentralization, as exemplified by Cisco's Fog Computing initiative [4] where the goal is to process data as close to the source as possible. This is motivated both by efficiency and by data privacy. We also believe that an important first step to achieve privacy is to avoid the collection of data in a central location, however, additional measures are required to avoid information leakage due to the publication of query results.

Here, we are concerned with the scenario in which each networked device stores only a small amount of data (typically collected locally), while there are

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2017)* in conjunction with the VLDB 2017 Conference in Munich, Germany. The proceedings of VLIoT@VLDB 2017 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

many participating devices in the network. This model covers a wide range of applications including smart metering [21], collaborative mobile platforms [20] and Internet of Things platforms [25].

We focus on stochastic gradient descent (SGD) as our learning algorithm of choice. This algorithm forms the basis of the decentralized gossip learning framework that supports various model fitting problems including linear models for classification and matrix factorization for recommender systems [13, 19]. In this framework, SGD visits data records with the help of a random walk over the network and updates a model approximation based on each record using the local gradient at that record. In general, SGD is a preferable method in large scale data mining [6] due to its scalability and simplicity. In our edge computing scenario the simplicity of SGD is a very important advantage: All the sensitive computations can be made strictly local to network nodes, as we will show later. In addition, no aggregation, synchronization, or central collection of data is necessary.

We work within the framework of differential privacy [10], where information leakage can be explicitly bounded by, for example, adding appropriately engineered noise to the query output. The standard assumption in this framework is that the database is stored at a central trusted location where secure computations can be carried out and only the end result of the computation is made public. Without differentially private mechanisms, even if all the computations are performed securely, the privacy of the data is still not guaranteed as the output of any secure computation might leak information about individual data records indirectly.

In our decentralized setup we must face much stricter privacy requirements than in the centralized case. Here, nodes form a network, in which adversarial nodes can also participate. For this reason any computations that are performed by the individual nodes must also be protected. In other words, we assume that every single node acts as a tiny private database on which queries are run and the final output is computed based on a large set of such public queries. This is because honest but curious adversaries can participate in the computation itself. They can, for example, surround any given node learning about its input and output thereby learning the end result of the local computation that is performed by the surrounded node. While it is possible to design protective measures against honest but curious adversaries, here we simply treat the end result of every local computation public. Our goal here is to design and evaluate decentralized differentially private variants of SGD learning in the context of linear models.

The differential privacy of model fitting via optimization and in particular SGD was investigated earlier in a number of publications. Generic frameworks, such as PINQ [12] and GUPT [17], have been proposed that do not readily lend themselves to secure distributed implementations. Chaudhuri et al. propose a method based on adding noise to the end result only, or perturbing the objective function itself [8, 9]. These approaches do not allow for an obvious secure distributed implementation either in our framework. In the case of output perturbation the local computations are not protected. In the case of objective function perturbation, in a decentralized system the nodes first have to agree on the secret noise term to be added to the objective function, which is not feasible if we assume that adversaries can participate in the computation.

Song et al. propose a method [23] that does allow a distributed implementation [14]. There, instead of perturbing the objective function, each update during the iterative gradient descent procedure is made differentially private. This allows for fully local gradient updates where the resulting gradient and the updated model can be made public. This prevents any uncontrolled data leakage as long as the personal computing device is not compromised. However, due to the theoretical properties of differential privacy, each data point can be accessed only a limited number of times, after which the data points have to be thrown away, that is, they cannot be accessed for any purpose without an increased risk of data leakage. This reduces the flexibility of the method.

Our novel contribution is proposing and evaluating novel differentially private linear algorithms that are based on every participating device publishing a perturbed version of the personal data it stores locally. Our SGD algorithm accesses only these public noisy data records and fits a linear model. Note that the resulting noisy model will also be protected by differential privacy since it is based on public noisy data only. We also evaluate differentially private mini-batch SGD implemented using gradient perturbation.

The data perturbation approach that we are proposing has not been investigated in the literature, most likely because it results in extremely noisy data and also because, as we will show, it is not ideal in the case when there are many records at every participating node, since in this case a mini-batch approach with gradient perturbation is superior. However, an important advantage of data perturbation in our setting is that—since these points can be accessed an arbitrary number of times—the fitted model can make use of all the available information in the noisy data, as opposed to the noisy gradient approaches proposed earlier. This is important because in a decentralized environment it is rather hard to guarantee that nodes are used only a limited small number of times while processing all the nodes in a

robust and efficient manner. With data perturbation we can apply known robust and efficient decentralized methods without any change, so we can achieve much faster convergence, as we will demonstrate.

In addition, the data perturbation approach will be demonstrated to offer similar or better performance than the gradient perturbation approach when every node has only one data record. However, the gradient perturbation method will be shown to produce better quality models if every node has more than one record, as expected. In this case, we can implement a mini-batch SGD algorithm that, with the same privacy budget, achieves a performance close to that of the noiseless learning algorithm depending on the mini-batch size.

## 2 BACKGROUND

### 2.1 Machine Learning

In this paper we focus on the sub-area of the machine learning called supervised learning or classification. Here, we are given a data set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of $n$ observations. The instances of the data set are composed of a feature vector $x \in R^d$ and the corresponding class label $y \in C$ where $d$ is the dimension of the problem and $C$ is the domain of the classes. For example, when $C = \{-1, 1\}$ we talk about binary classification. The goal of machine learning is to find a function $f_w : R^d \rightarrow C$ that can classify the given data instances correctly, and also those instances that are not presented in the data set. This latter property is called generalization.

Moreover, in some cases the number of classes is more than two (e.g. $K$), we talk about multi-class classification. But these problems can be imagined and handled as $K$ binary classification problems [3].

### 2.2 Stochastic Gradient Descent

For a model $f_w()$ that depends on parameters $w$ and for a loss function $\ell()$ that defines the error between the prediction and the actual class, the learning problem we described above can be expressed as the optimization problem

$$w = \arg\min_w J(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_w(x_i), y_i) + \frac{\lambda}{2}\|w\|^2, \tag{1}$$

where $(\lambda/2)\|w\|^2$ is the regularization term with parameter $\lambda$. The regularization term helps the model to avoid overfitting the data set, thus helping generalization.

Gradient descent (GD) is an optimization method that can solve the optimization problem above. The parameter vector is updated iteratively by the gradient of the objective function:

$$\begin{aligned} w_{t+1} &= w_t - \eta_t(\frac{\partial J}{\partial w}) \\ &= w_t - \eta_t(\lambda w + \frac{1}{n}\sum_{i=1}^{n} \nabla\ell(f_w(x_i), y_i)), \end{aligned} \tag{2}$$

where $\eta_t$ is the learning rate at time $t$ that scales the size of the gradient step.

Stochastic gradient descent (SGD), on the other hand, computes the gradient on one instance only in every step. The update rule becomes

$$w_{t+1} = w_t - \eta_t(\lambda w + \nabla\ell(f_w(x_i), y_i)). \tag{3}$$

Here two restrictions have to be applied on the learning rate to guarantee the convergence of the algorithm, namely $\sum_t \eta_t^2 < \infty$ and $\sum_t \eta_t = \infty$ [5].

### 2.3 Differential Privacy

Differential privacy [10] is concerned with the leakage of personal information due to publishing the results of a given query over a database. Even if performed securely, the result of a query can leak information about individual records, for example, the maximum of a set of values is an individual record in itself. Differential privacy is achieved if noise is added to the query result in such a way that the following definition is satisfied.

**Definition 1 (Differential Privacy):** *A randomized query $F : \mathcal{D} \mapsto \mathbb{R}^d$ is* $\epsilon$-differentially private *iff*

$$\forall x : e^{-\epsilon} \leq \frac{P(F(D) = x)}{P(F(D') = x)} \leq e^{\epsilon} \tag{4}$$

*for all pairs of databases $D$ and $D'$ that differ in at most one record, where $\mathcal{D}$ is the set of possible databases.*

That is, if we change one element in the database, the same output should be expected with a probability close to that over the original database. This way, one record never "matters too much" thereby limiting the information leakage as a result of the query.

A randomized query typically means adding noise to an otherwise deterministic query. This added noise is designed specifically for a given query and parameter $\epsilon$ such that the definition of $\epsilon-$differential privacy is satisfied. In more detail, to generate the additive noise we need to pick a noise distribution and the right distribution parameters. A common approach to take is to first determine the so-called *sensitivity* of the query [10, 11]:

**Definition 2 (Global Sensitivity):** *The* global $L^1$-sensitivity $Z_F$ *of $F$ is given by*

$$Z_F = \max_{D,D' \text{ differ in one record}} \|F(D) - F(D')\|_1, \tag{5}$$

*where $\|\cdot\|_1$ is the $L^1$ norm.*

The definition can be generalized by replacing the $L^1$ norm with a different norm. The usual norms to apply are the $L^1$ norm and the $L^2$ norm. In the case of applying the $L^1$ norm, the following noise distribution can be used: we need to add to all the dimensions of the output independent noise drawn from $\mathrm{Laplace}(0, Z/\epsilon)$ (where $Z$ is the global sensitivity of the query), which will result in $\epsilon-$differential privacy. Based on the theoretical results described in [11], noise can be generated for any other norms.

## 2.4 Support Vector Machines and Logistic Regression

**Support Vector Machines:** The linear support vector machine (SVM) is an approach to classification in machine learning that looks for the hyperplane that separates the classes from each other and has the maximal margin (the distance between the hyperplane and the training instances). Several approaches are known for solving the SVM problem using a stochastic gradient approach. The most appealing algorithm we are aware of is Pegasos [22], a sub-gradient approach (since the SVM loss function is non-differentiable, sub-gradients are used instead of gradients).

$$\ell(f_w(x), y)) = \max(0, 1 - w^T(yx)) \tag{6}$$

The sub-gradient that is used in Pegasos at the instance $(x, y)$ with $y \in \{-1, 1\}$ can be formulated as

$$\nabla_t = \lambda w_t + \mathbb{1}[w^T(yx) < 1]yx, \tag{7}$$

where $\mathbb{1}$ is a function that returns the value one if the condition is true, otherwise it returns zero. Parameter $\lambda$ is the regularization parameter. The update rule is $w_{t+1} = w_t - \eta_t \nabla_t$ where the learning rate $\eta_t = 1/(\lambda t)$. This gives

$$w_{t+1} = (1 - \frac{1}{t})w_t + \eta_t \mathbb{1}[w^T(yx) < 1]yx. \tag{8}$$

**Logistic Regression:** In the case of logistic regression [16] the optimization problem is expressed as a maximization problem, since it is more natural to think of it as maximizing the logarithm of the likelihood.

$$\ell(f_w(x), y)) = ln(1 + exp(-w^T(yx))) \tag{9}$$

The gradient that can be used to update the parameters of the model is

$$\nabla_t = \lambda w_t + (1 - \frac{1}{1 + exp(-w^T yx)})yx \tag{10}$$

**Algorithm 1** Gossip Learning Framework

```
1: (x, y) ← local training example
2: currentModel ← initModel()
3: loop
4:     wait(Δ)
5:     p ← selectPeer()
6:     send currentModel to p
7: end loop
8: procedure ONRECEIVEMODEL(w)
9:     w ←updateModel(w, x, y)
10:    currentModel ← (w+ currentModel )/2
11: end procedure
```

And the update rule is given as

$$w_{t+1} = (1-\frac{1}{t})w_t + \eta_t(1-\frac{1}{1 + exp(-w^T yx)})yx \tag{11}$$

These update rules will form the basis of our differentially private algorithm.

## 2.5 Distributed Machine Learning

In our system model we are given a network of a large number of computational units (e.g. PCs, smart phones, tablets, wearable units, or smart meters). The members of this network can communicate with each other by message passing. A node in this network can send a message to another node whose address is known locally. We assume that every node in this network has only one training example $(x, y)$, but we can benefit from having more local data. The set of these isolated examples form our machine learning database. We would like to learn a model over these instances in a fully distributed manner while also preserving privacy.

The Gossip Learning Framework [19] is a possible way to learn models in this fully distributed environment. The basic idea is that in the network many models perform random walks and are updated at every node using the local example. The number of walking models is in the range of the number of nodes in the network. These walks are continuously averaged so that in effect a parallel SGD is approximated. In more detail, every node executes Algorithm 1. A node in the network first initializes a local model, then iteratively sends its local model to a randomly selected node in the network. The address of the randomly selected node is provided by a peer sampling service (e.g. the NewsCast [24] protocol). When a node receives a model, it updates the model by its locally stored training example using the SGD update rule, and then stores the updated model as its local model after averaging it with the previously received model. Using this protocol the models stored by the nodes will converge to the same global optimum.

Our study is inspired by gossip learning in the sense that we focus on SGD algorithms that are implemented through a random walk of the evolving model over the network. We will assume that this random walk itself is secure. Ideas for achieving secure random walks were outlined, for example, in [2]. Here, we focus on privacy. In order to achieve privacy, we will apply a differentially private variant of the local update step.

## 3 ALGORITHM

Our approach is based on publishing a noisy version of the local data at each node. After this, any algorithm can perform any amount of processing as long as the original noise free data is no longer accessed.

First, let us make the observation that the update rules in equations (8) and (11) depend only on $y \cdot x$ and the values of $x$ and $y$ are never used separately. Based on this, it suffices to publish noisy versions of $xy$.

We publish these values so that the entire set of values is $\epsilon-$differentially private for some given $\epsilon$. Since all the published query results belong to non-overlapping subsets of the database (each containing only one data instance) it is enough to make sure that every node publishes its $yx$ value with $\epsilon-$differential privacy. We will calculate the required amount of noise based on the sensitivity of $xy$ as defined in Section 2.3.

It is easy to see that for a given norm $\|\cdot\|$ the global sensitivity of $xy$ is $2\max_x \|x\|$. Given that for any different instances $(x, y)$ and $(x', y')$, using that $|y| = 1$, we have

$$\|yx - y'x'\| \leq \|yx\| + \|-y'x'\| = \|x\| + \|x'\| \leq 2\max_x \|x\|, \tag{12}$$

the global sensitivity of $xy$ is given by

$$Z_{yx} = \max_{(x,y)\neq(x',y')} \|yx - y'x'\| \leq 2\max_x \|x\|. \tag{13}$$

The value of $\max_x \|x\|$ is a global query itself. However, we can apply local normalization at all the nodes, that is, we can normalize each instance $x$ so that $\|x\| = 1$. This directly gives $Z_{yx} = 2$. Especially in higher dimensional spaces where the norms of the vectors tend to be similar, this does not introduce too much damage to the structure of the data. We repeat that, as explained in Section 2.3, any suitable norm can be chosen. The chosen norm in turn defines the formula for the noise term, and the corresponding normalization can be applied.

To sum up, all the nodes $i$ publish the value of $y_i x_i + N_i$, where $N_i$ is the noise term calculated as explained above. The database $\{y_1 x_1 + N_1, \ldots, y_n x_n + N_n\}$ can freely be post-processed by any algorithms. For the Pegasos algorithm, the update rule will be

$$w_{t+1} = (1 - \frac{1}{t})w_t + \eta_t \mathbb{1}[w^T(yx + N) < 1](yx + N), \tag{14}$$

while for logistic regression the update rule is

$$w_{t+1} = (1 - \frac{1}{t})w_t + \eta_t(1 - \frac{1}{1 + exp(-w^T(yx + N))})(yx + N). \tag{15}$$

The output of these algorithms will inherit the property of $\epsilon-$differential privacy. In our approach, the $y_i x_i + N_i$ values stay at node $i$ locally, and processed by gossip learning, but they could just as well be collected centrally without posing privacy problems.

## 4 EXPERIMENTS

A key intuition regarding this approach is that, although the amount of noise is rather substantial (in the order of the raw data itself), if the database is large enough then the structure of the data will still be preserved, also considering that the linear algorithms we apply are based on a structurally simple model (a hyperplane).

### 4.1 Datasets

In our experiments we used datasets from the UCI [1] machine learning repository and an artificial dataset as well. The real databases include the Mnist dataset, where the task is to recognize digits from 0 to 9 based on images of 28×28 gray intensity pixels; the Segmentation dataset, where connected image parts have to be identified; the Spambase dataset, where emails have to be categorized based on higher level feature representations. The artificial dataset is a set of two dimensional points belonging to two classes, as illustrated in Figure 2. The classes are generated from Gaussian distributions that have different expected values but have the same deviation, i.e. N($\mu_1$,$\sigma$) and N($\mu_2$,$\sigma$) for the negative and the positive classes, respectively, where $\mu_1 = (0, 10)$, $\mu_2 = (10, 0)$ and $\sigma = 1$ along both dimensions. The main properties of the above mentioned datasets can be seen in Table 1. The datasets represent various machine learning problems and have a variety of feature and sample sizes and number of classes.

In our evaluations we built a model on the training set of the dataset in question and evaluated this model on a non-overlapping test set. The evaluation metric we used is the accuracy of the prediction performance, that is, the fraction of correctly classified data instances on the test sets.

**Table 1: The main properties of the datasets**

| | random | Segmentation | Spambase | MNIST |
|---|---|---|---|---|
| **Training set size** | 20 000 | 2 310 | 4 140 | 60 000 |
| **Test set size** | 2 000 | 210 | 461 | 10 000 |
| **Number of features** | 2 | 19 | 57 | 784 |
| **Number of classes** | 2 | 7 | 2 | 10 |
| **Class-label distribution** | uniform | uniform | 6:4 | uniform |

We normalized the datasets by transforming each feature independently into the [0,1] interval. This process is done for a feature $i$ by applying

$$\hat{f}_i = \frac{f_i - min_i}{max_i - min_i},$$

where the $max_i$ and $min_i$ are the maximal and minimal value of the feature on the training set. After this normalization we scaled the length of the instances to 1 according to the $L^1$ vector norm. The length scaling is performed locally.

Note that some of the datasets contain more than two classes. In these cases we teach a binary classifier for each class, where the positive cases belong to the given class and the negative cases are formed by the rest of the classes. This also means that we have more than one binary classification problems that have to share the available privacy budget. If we are given a budget of $\epsilon$ and we have $c$ binary classifiers then we assign a budget of $\epsilon/c$ to each classifier.

## 4.2 The Drawbacks of Gradient Perturbation

In our first set of experiments we reproduce our own previous results [15] using the $L^1$ norm. Here, instead of the data instances, the calculated update gradient is perturbed in each update step. For this reason, only a limited number of updates are allowed for each data instance. Each training sample has a privacy budget $\epsilon = 50$ that can be managed in a number of ways. One can, for instance, set a finite number of $k$ allowed updates and use $\epsilon/k$ for each one. This means multiplying the magnitude of the noise term by $k$ for each update. In the experimental evaluation we study this parameter looking at the cases of $k = 1$ and $k = 5$. We can also follow a different approach and divide $\epsilon$ into an infinite number of parts by using $\epsilon/2^t$ for update $t$. This way noise increases exponentially, but we can execute as many updates as we wish using the same example.

The reproduced results are shown in Figure 1. As the figure shows the drawback of this method is that due to the limited budget the algorithm might never reach convergence. There are two possible ways of obtaining an unlimited number of update steps that are required for the convergence of SGD. One way is when we have an unlimited number of training examples, e.g. the instances of the dataset can be sampled from a parametric distribution. And the other is adding noise to the input data directly, the method we described here previously.

## 4.3 Results

We first experimented with our artificial dataset to demonstrate the main intuition behind adding noise to the data as opposed to the gradient. As we mentioned above, the additional noise preserves the direction of the hyperplane that separates the instances of different classes. To demonstrate this property we applied $L^1$ noise with different privacy levels on our artificial dataset and plotted the results in Figure 2. Note that the figure represents the points $x_i+N_i$ for all records $i$ (where $N_i$ is the noise term) instead of $y_i x_i + N_i$. This visualization is correct because $y_i x_i + N_i = y_i(x_i + N_i/y_i)$ where $N_i/y_i$ has the same distribution as $N_i$ since $y_i \in \{-1, 1\}$ and the distribution of $N_i$ is symmetric with a center of zero. This also implies that we can think of the privacy mechanism as adding noise to $x_i$.

We then used the LibSVM [7] and the PegasosSVM [22] software packages to classify the instances and based on the result we reconstructed the separating hyperplane as shown in Figure 2. In this illustrating example, the model produced by the SVM solver remains exactly the same in expectation. This is because the arrangement of the original classes and the distribution of the noise are such that the distribution of the original classes are mirror images of each other over the separating hyperplane and this property is preserved in the case of the noisy classes as well.

This assumption will not hold in general but this example illustrates the intuition behind the approach nevertheless: Even a very large amount of noise can

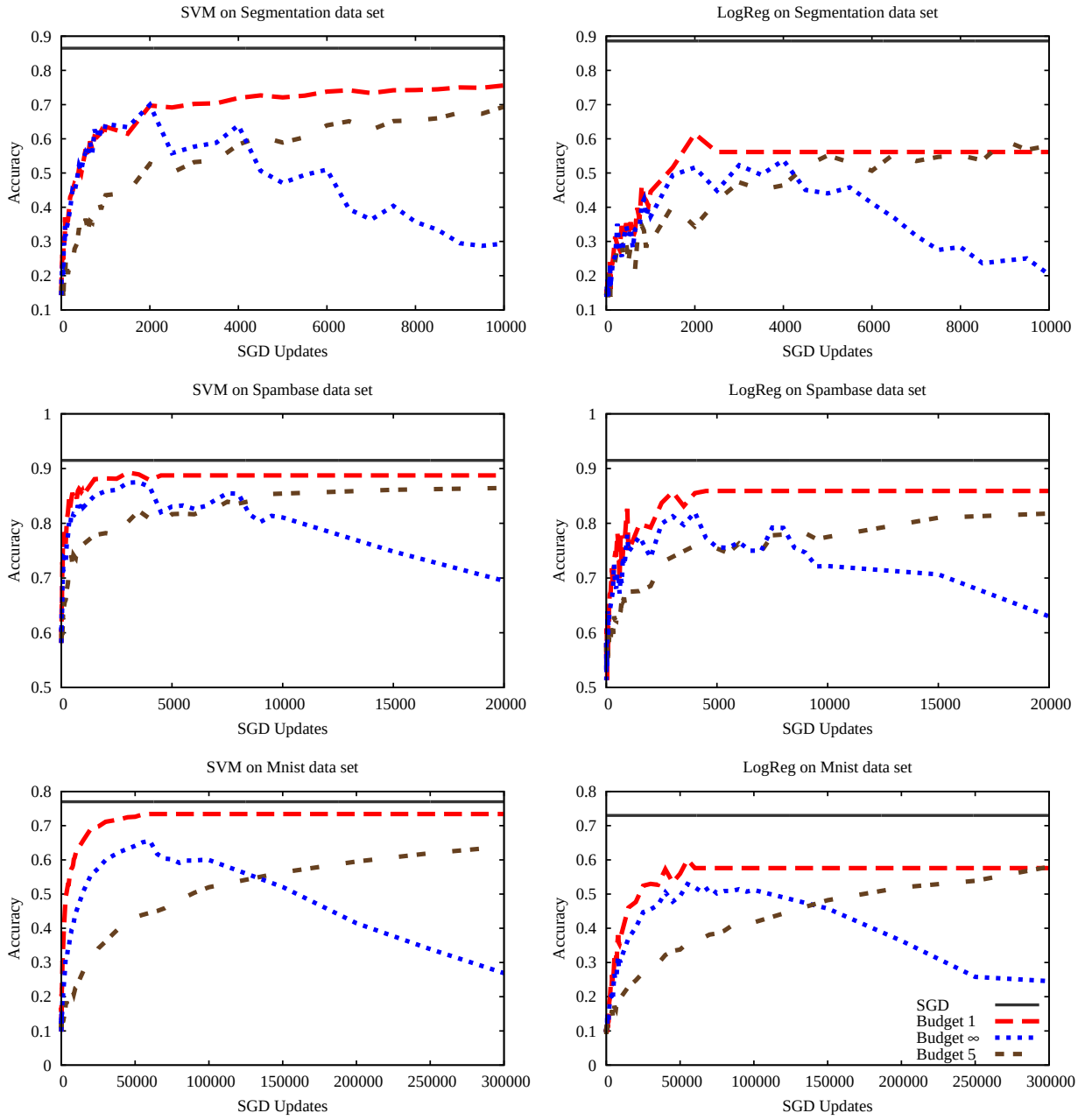



**Figure 1: The effect of using different strategies for privacy budget management for gradient perturbation**

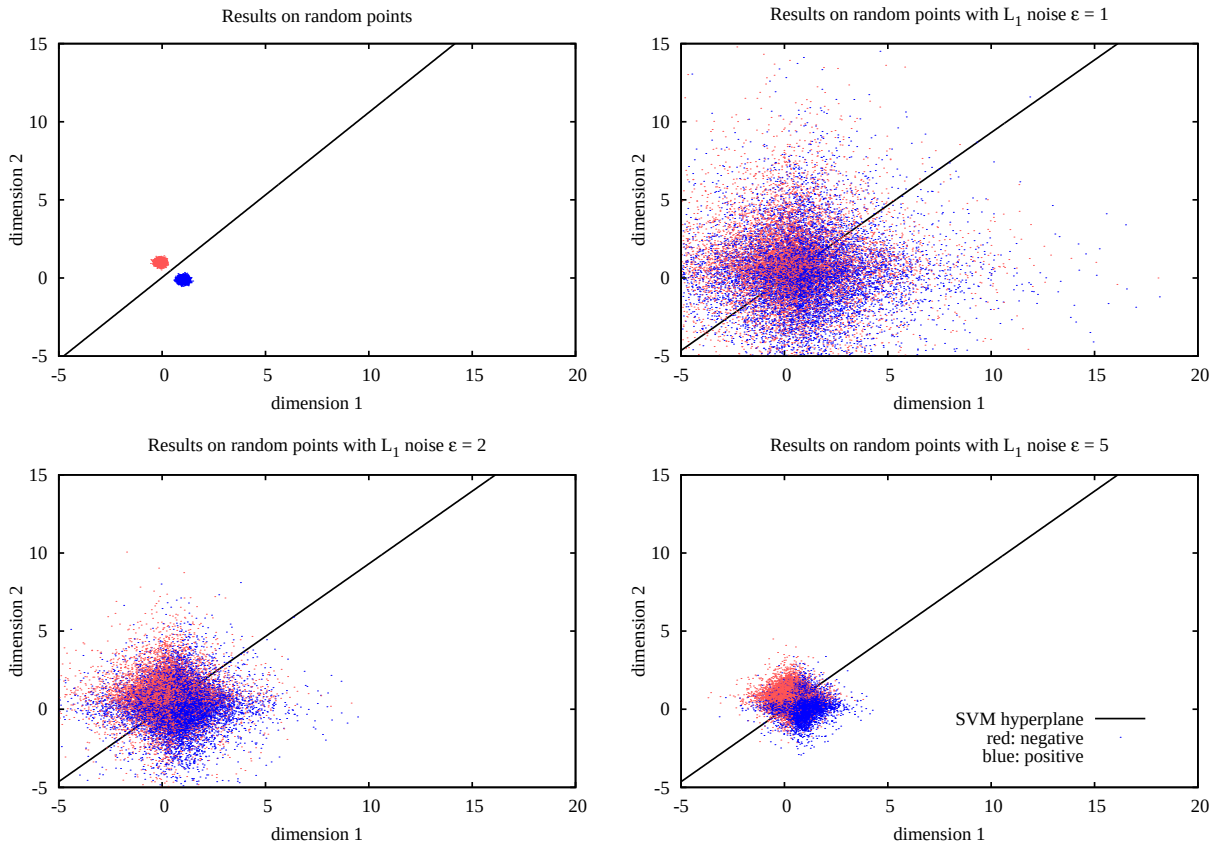



**Figure 2: The optimal hyperplane of the linear SVM on the artificial dataset with various privacy levels.**

leave the optimal plane completely unchanged thus, in the best case, we could get completely accurate results even with very large amounts of noise.

We now present our empirical results illustrating the performance of data perturbation and gradient perturbation. Every result we present is the average of 10 independent runs. We vary the privacy levels, and we also apply the mini batch technique. In the distributed setting, the mini batch technique assumes that a single node has $k > 1$ examples. This also means that the number of nodes will be the number of training examples divided by $k$. In this case, the node can compute a combined gradient, namely the average of the gradients of the data samples it has. In the case of gradient perturbation, the noise has to be added only to this combined gradient.

In the mini-batch gradient perturbation algorithm we have many choices to manage the privacy budget $\epsilon$, similarly to the cases explained in Section 4.2. The first variant we look at is using all the budget of $\epsilon$ at once on a node with $k$ samples, thus we have one update per node. This will result in less noise as a function of $k$, given that the sensitivity of the average query (see Section 2.3) in the case of batch size $k$ can be easily shown to be $1/k$ times the sensitivity of a single data record. The results are shown if Figure 3.

The second variant we look at is aimed at keeping the number of update steps (that is, the number of gradients used) constant and equal to the number of data records. This means that we visit each mini-batch of size $k$ exactly $k$ times. Each time, we use a budget of $\epsilon/k$. The results are shown if Figure 4.

In the case of data perturbation, we perturb all the individual data points, so the mini batch technique will not affect the final theoretical quality of the model. The mini batch technique still offers a practical improvement though: It improves convergence speed since the gradients used to update the model will be smoother. However, now we are interested only in the quality of the final model, and optimize the model until convergence no matter how many gradient updates it takes, so batch size is irrelevant for this experiment.

In our experiments we varied the privacy level $\epsilon$ and batch size and ran experiments with all combinations. The batch size and the privacy level $\epsilon$ both took values from the set $\{1, 5, 10, 50, 100\}$ and we apply heat-maps

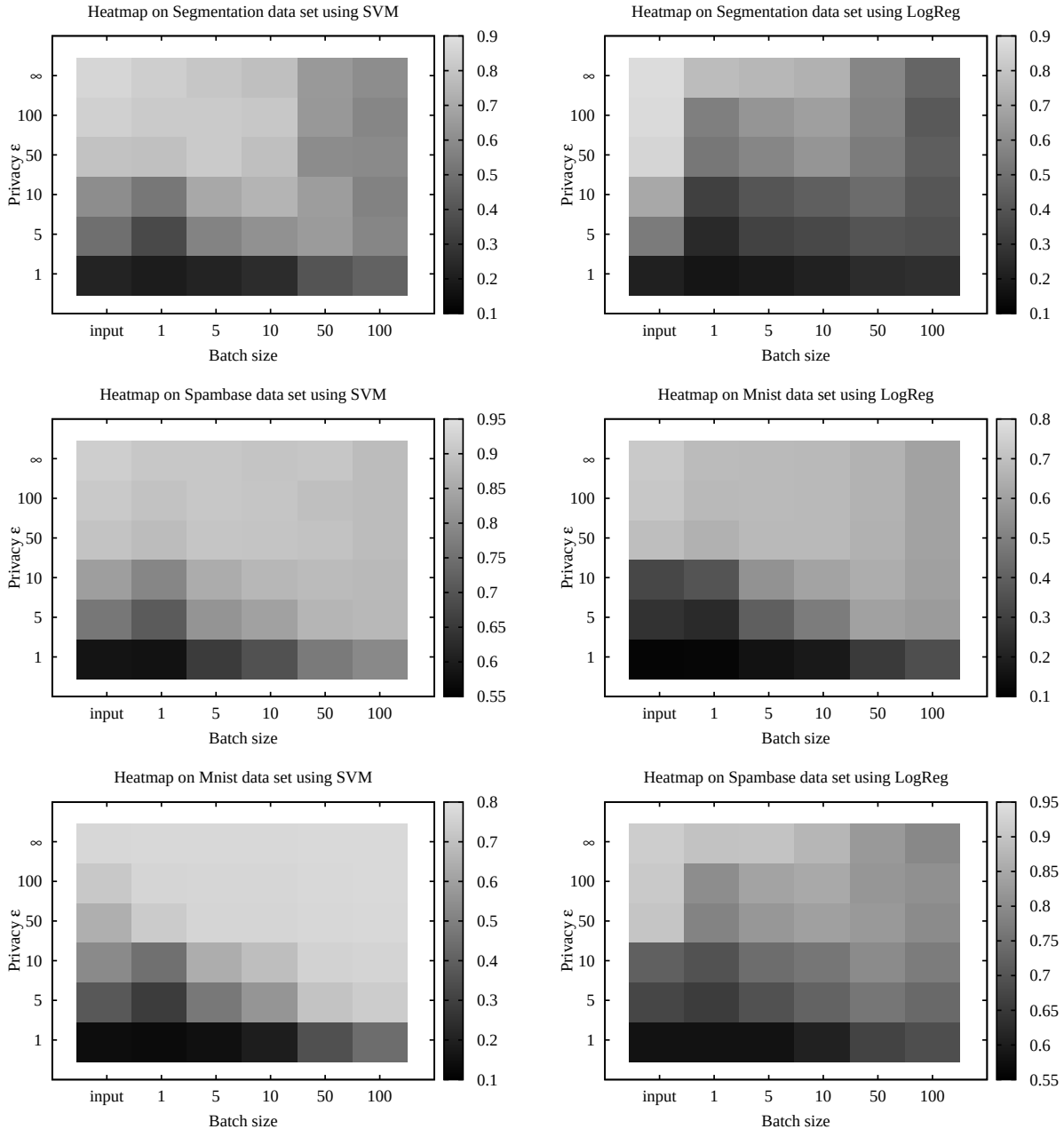



**Figure 3: Experiments with various batch sizes and privacy levels. The top row and left column of each heat-map represent the results without privacy preservation and the results applying data perturbation, respectively. Lighter color means better performance.**

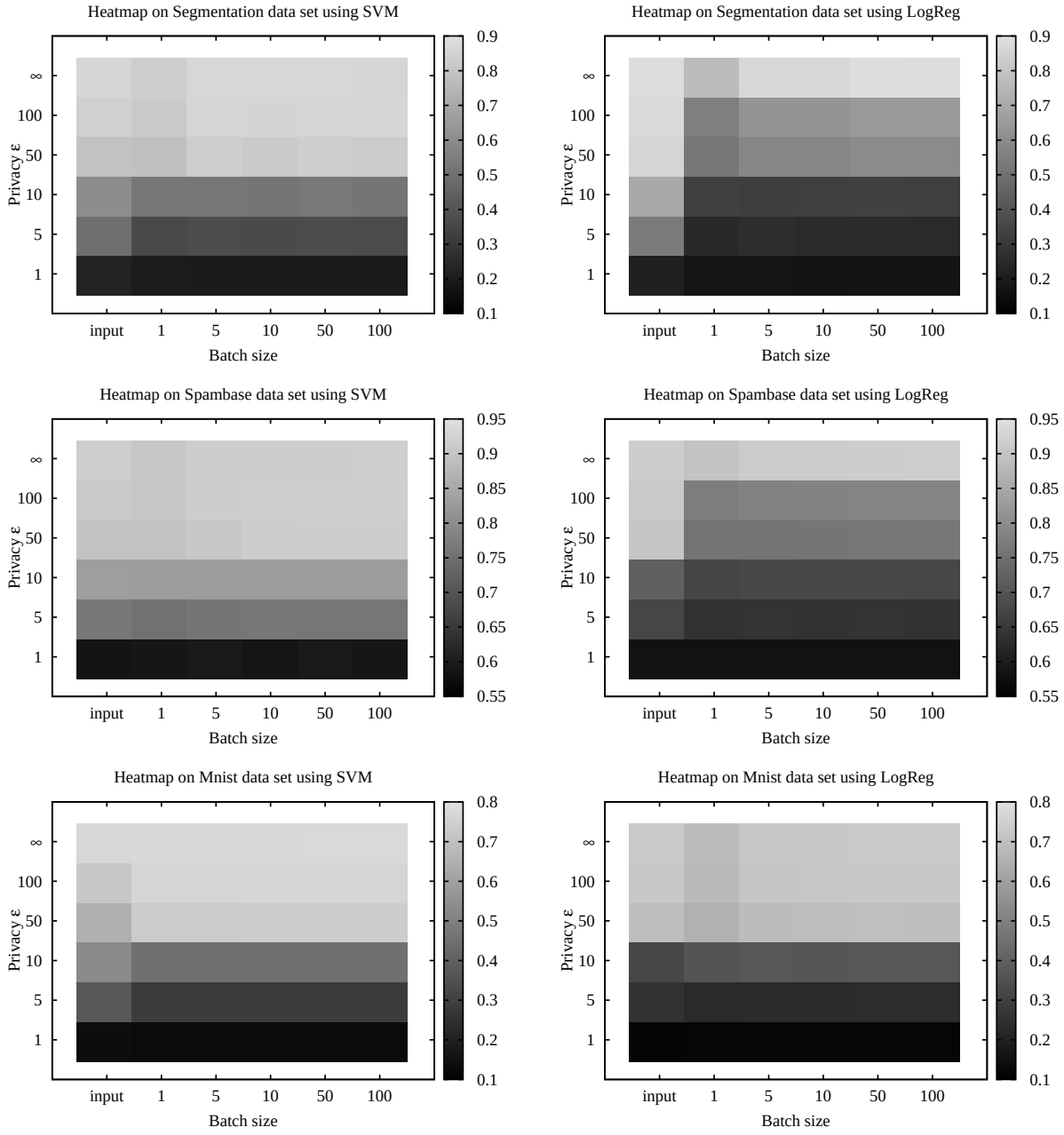



**Figure 4: Experiments with various batch sizes and privacy levels, allowing an equal number of gradient updates in each experiment. The top row and left column of each heat-map represent the results without privacy preservation and the results applying data perturbation, respectively. Lighter color means better performance.**

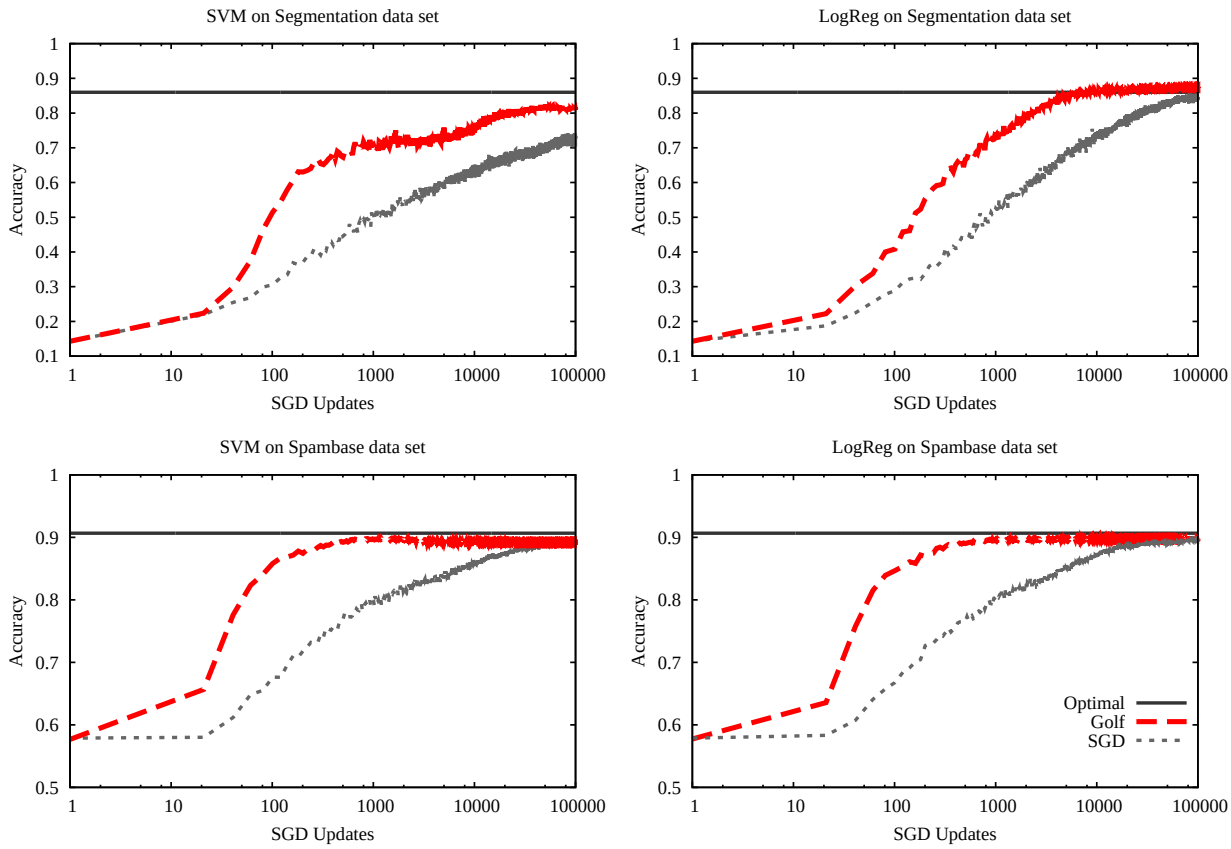



**Figure 5: Simulation results with Golf in a fully distributed environment**

to illustrate all the combinations in Figures 3 and 4. In addition, the top row of the heat-maps represents the results of the data perturbation method, and the left column shows the results of the algorithms without any privacy preservation (that is, with $\epsilon = \infty$).

For the optimization the algorithms described in Section 2.4 were used. The learning rate parameter was $\lambda = 10^{-4}$ for both logistic regression and Pegasos SVM. No parameter optimization was performed.

As expected, lower privacy levels (i.e., higher $\epsilon$ values) result in better performance. That is, for a fixed batch-size the values in the rows getting better and better towards the upper parts of the figures. The heat-maps also suggest that the logistic regression learning algorithm tolerates the noisy data less than the SVM algorithm. We note here though that no parameter search was performed, the algorithms were run with the same default learning rate.

The optimal batch-size depends on the number of nodes in the variants where the number of updates depends on the number of nodes (Figure 3). With a larger database, larger and larger batch sizes are optimal. When the number of gradient updates is fixed, the performance does not seem to depend on the batch size (Figure 4). The likely explanation is that although the mini batch method does offer a smoother gradient, this smoothing is not significant compared to the amount of noise we add to the gradient to preserve privacy. In any case, when enough data is available locally, it appears to be more advisable to run gradient perturbation with a suitable batch size.

However, with $k = 1$ it is preferable to apply data perturbation, as illustrated by the results. We should not forget that data perturbation has the additional benefit of being reusable in any learning problem or algorithm, whereas after applying the gradient perturbation algorithm, the data cannot be used again.

The last set of experiments we present were performed to test the algorithms in a fully distributed environment with additional optimizations that are applicable in such environments. Through these experiments we can illustrate another advantage of data perturbation, apart from the re-usability of the datasets. Namely, the data perturbation approach allows us to apply the Gossip Learning Framework (Golf), introduced in Section 2.5. Note that Golf is impossible to implement with gradient

perturbation, because in Golf every node performs an update in every cycle, whereas with gradient perturbation we can implement only a single random walk in the entire system after which we must throw away the data.

We used the PegasosSVM and Logistic Regression learning algorithms to implement SGD and we used the PeerSim simulation environment [18] to simulate Golf. We set $\epsilon = 50$. The number of nodes in the network was the same as the number of training instances. Each node has only one training sample and at the beginning of the simulation each node initialized a learning model. We evaluated the performance of the models over a sample of 100 nodes in the network, chosen uniformly at random in every gossip cycle.

The results are shown in Figure 5. As expected, all variants converge to the theoretical maximum performance. The convergence speed is different, though. Golf is clearly the fastest. Please note the logarithmic scale on the time axis. In fact, Golf offers a radically faster convergence rate due to the implemented asynchronous parallel SGD compared to the sequential SGD. Again, this is possible only because data perturbation (unlike gradient perturbation) allows us to apply Golf directly.

## 5 Conclusions

We proposed and evaluated the data perturbation method for supporting stochastic gradient descent for linear models. The method provides explicitly controlled differential privacy for the end result of stochastic gradient descent search and it allows the entire dataset to be published as well. This way, the learning data can be shared and used in an unlimited number of times. This property allows us to learn more models, fine tune their parameters, run the SGD algorithm until convergence, and run off-the-shelf distributed learning algorithms like Golf without change.

We demonstrated that the achieved performance of the models are better than the results of gradient perturbation if every node has only a single record. If every node has a large number of data records, mini-batch gradient perturbation results in a better performance with the same privacy budget along with the appropriate mini-batch size. However, with gradient perturbation, the data can no longer be used for any other queries.

## Acknowledgements

This research was supported by the Hungarian Government and the European Regional Development Fund under the grant number GINOP-2.3.2-15-2016-00037 (Internet of Living Things).

## References

[1] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[2] K. Birman, M. Jelasity, R. Kleinberg, and E. Tremel, "Building a secure and privacy-preserving smart grid," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 131–136, January 2015.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, New York, NY, USA, 2012, pp. 13–16.

[5] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Springer Berlin Heidelberg, 2012, vol. 7700, pp. 421–436.

[6] L. Bottou and Y. LeCun, "Large scale online learning," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004.

[7] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011.

[8] K. Chaudhuri and C. Monteleoni, "Privacy-preserving logistic regression," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Curran Associates, Inc., 2008, pp. 289–296.

[9] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *J. Mach. Learn. Res.*, vol. 12, pp. 1069–1109, July 2011.

[10] C. Dwork, "A firm foundation for private data analysis," *Commun. ACM*, vol. 54, no. 1, pp. 86–95, January 2011.

[11] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, S. Halevi and T. Rabin, Eds. Springer Berlin Heidelberg, 2006, vol. 3876, pp. 265–284.

[12] A. Friedman and A. Schuster, "Data mining with differential privacy," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'10)*, New York, NY, USA, 2010, pp. 493–502.

[13] I. Hegedűs, Á. Berta, L. Kocsis, A. A. Benczúr, and M. Jelasity, "Robust decentralized low-rank matrix decomposition," *ACM Transactions on Intelligent Systems and Technology*, vol. 7, no. 4, pp. 62:1–62:24, May 2016.

[14] I. Hegedus, Á. Berta, and M. Jelasity, "Robust decentralized differentially private stochastic gradient descent," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 7, no. 2, pp. 20–40, 2016.

[15] I. Hegedűs and M. Jelasity, "Distributed differentially private stochastic gradient descent: An empirical study," in *Proceedings of the 24th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'16)*. Heraklion, Greece: IEEE Computer Society, 2016, pp. 566–573.

[16] T. M. Mitchell, *Machine Learning*, 2nd ed., E. M. Munson, Ed. New York: McGraw-Hill, 1997.

[17] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler, "Gupt: privacy preserving data analysis made easy," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2012, pp. 349–360.

[18] A. Montresor and M. Jelasity, "Peersim: A scalable P2P simulator," in *Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing*, Seattle, Washington, USA, Sep. 2009, pp. 99–100.

[19] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 556–571, 2013.

[20] A. S. Pentland, "Society's nervous system: Building effective government, energy, and public health systems," *Computer*, vol. 45, no. 1, pp. 31–38, January 2012.

[21] A. Rial and G. Danezis, "Privacy-preserving smart metering," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society (WPES'11)*. New York, NY, USA: ACM, 2011, pp. 49–60.

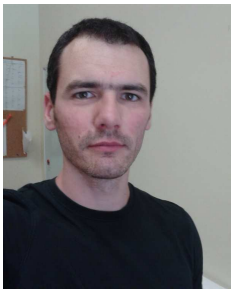
[22] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "Pegasos: primal estimated sub-gradient solver for SVM," *Mathematical Programming B*, 2010.

[23] S. Song, K. Chaudhuri, and A. D. Sarwate, "Stochastic gradient descent with differentially private updates," in *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, December 2013, pp. 245–248.

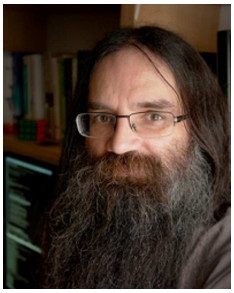
[24] N. Tölgyesi and M. Jelasity, "Adaptive peer sampling with newscast," in *Euro-Par 2009*, ser. Lecture Notes in Computer Science, H. Sips, D. Epema, and H.-X. Lin, Eds., vol. 5704. Springer-Verlag, 2009, pp. 523–534.

[25] C.-W. Tsai, C.-F. Lai, M.-C. Chiang, and L. Yang, "Data mining for internet of things: A survey," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 1, pp. 77–97, 2014.

## AUTHOR BIOGRAPHY



**István Hegedűs** is a research assistant at the University of Szeged. He obtained his PhD degree from the University of Szeged in 2017, under the supervision of M. Jelasity. His main research interests are fully distributed algorithms and machine learning.



**Márk Jelasity** is a full professor at the University of Szeged. He obtained his PhD degree from the University of Leiden in 2001, and his DSc degree from the Hungarian Academy of Sciences in 2015. His research interests include decentralized algorithms for data aggregation and mining in large scale unreliable systems, data privacy, and self-organizing systems.