Split Learning for Finance: Comparing Architectures for Privacy-Aware Distributed Deep Learning

by
Björn Keyser

Professor A. van Halteren, Advisor C. Allaart

A thesis submitted in partial fulfillment
of the requirements for the
Degree of Bachelor of Science
in Computer Science

VRIJE UNIVERSITEIT
Amsterdam, the Netherlands
July 14, 2021

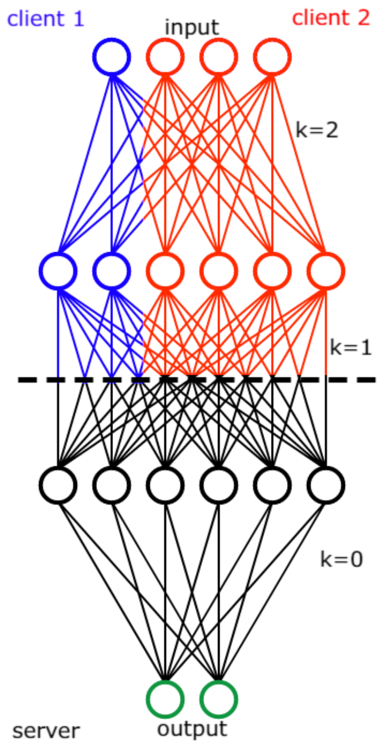
Abstract

UPDATED—July 14, 2021. Due to the sensitive nature of financial data - e.g. customer data or financial transactions - as well as due to the financial information sector being highly regulated, it is a necessity to keep this data private and unexposed. This becomes a challenge however, when some parties having different data about the same instances, want to collaborate by pooling their data and applying deep learning algorithms on this pooled data. This paper evaluates the performance of various configurations of a recently introduced privacy-aware deep learning technique called split neural network (split NN [1]) on certain financial scenarios which could benefit from using aggregated data, such as fraud detection and insurance claim approval. We show what factors are important when choosing the split network's hyperparameters, and what configurations can have detrimental effects on the split network's performance.

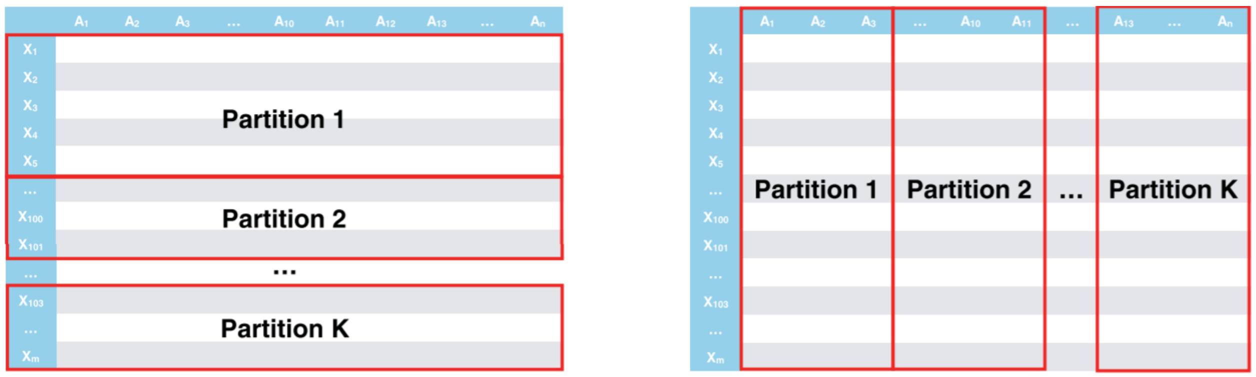
Introduction

Increasingly, deep learning algorithms are utilized for various tasks in the financial industry, such as detecting fraud, insurance approval, risk assessment and more. The accuracy of such tasks depends highly on the size of the data set used, as well as which variables are used to train on. Being able to aggregate different resources of data for this task, i.e. distributed deep learning, is thus of substantial utility and a vast area of research. In this paper, we focus on a way of combining resources' data to increase the amount of variables (or columns) on data, as opposed to combining data to increase the amount of instances (or rows). As a concrete example, one data source X1 has some credit card transaction data of John Doe, and another data source X2 has data on John Doe's birthplace, education level, occupation, number of children, etc., and these can now be pooled to make better predictions on some Y. For the visual thinkers, this data is said to be *vertically* split among X1 and X2 (see Figure 2a), as opposed to *horizontally* split, where the data set is split such that e.g. surnames from A-M are at X1 and N-Z are at X2, which usually only occurs within the same organization. Vertically split learning is useful when different entities want to collaborate and get the most out of their data.

Figure 1: A vertical split learning situation; in this case, Client 1 holds 1 feature and Client 2 holds 3 features over common entities.



Data collaboration in finance between companies is heavily hindered by a multitude of confidentiality related factors, such as data sharing regulations and lack of users'



(a) Horizontally partitioned data

(b) Vertically partitioned data

Figure 2: The difference between horizontally and vertically partitioning data. Rows refer to instances and columns refer to features. [2]

consent, as well as resource-constraints, all leading to uneasy cooperation between parties. That is, until privacy-aware distributed deep learning methods such as split NN were introduced [1], eliminating the need for collaborating entities to share raw data in order to train on it. Instead, split NN allows data holders to train on their data locally, and only share the activations of a certain chosen layer - the *cut* layer - when propagating forward and backward (these are the activations travelling from the second to the third layer in Figure 1, also called *smashed data*). Additionally, this solution requires minimal bandwidth in comparison to other private distributed deep learning methods; especially in settings with a large number of clients it is by far the most resource and communication efficient [3].

Lack of Literature

The applications of this technique are widespread among e.g. health, biomedicine and finance, but the versatile nature of this method also means that there are many possible configurations. However, despite there being various design choices and considerations to be made when deploying such a vertically split neural network, there is limited research on how to choose the optimal hyperparameters, e.g. what layer to choose as the cut layer, and how to distribute the features among entities. Optimal in this case regards the model's classifying performance; we will be evaluating F1 performance.

The Objective

The aim of our research is thus to compare and evaluate architectural configurations for three specific practical deep learning tasks regarding sensitive financial data, with our research question being; *how do different configurations affect a vertically partitioned split NN model's predictive performance?* More specifically, we intend to find *how varying the cut layer affects predictive performance*, as well as find out *how varying the feature distribution among two* *partitions affects the predictive performance*, and ideally find a way to generalize this. By iterating over each possible configuration and evaluating the split model's performance, we intend to find a rationale behind choosing the best architecture for specific challenges in need of distributed deep learning. Additionally, we will examine the effects of recombining the features among two partitions on the split model's performance, and propose a relation between a feature set's inter-correlations and its predictive performance.

Related Work

Besides split learning, there are currently a multitude of other privacy-aware distributed deep learning methods available, such as federated learning [4] , large batch synchronous stochastic gradient descent (SGD) [5] and more. These can each benefit from novel combinations with generic methods of protection such as differential privacy, homomorphic encryption and secure multi-party computation. An important distinction between the three private distributed deep learning methods, is that federated learning and large batch synchronous SGD share the full architecture and (hyper)parameters between the client and the server, along with intermediate representations of the model including gradients, activations and weight updates. There has been research showing that, although raw data is never shared in these methods, it can be reconstructed up to an approximation by adversaries, by utilizing the fact that architecture and parameters are not kept private. Split NN, on the other hand, does not share the architecture and weights, rendering the raw data less prone to reconstruction, whilst also minimizing the bandwidth required to share intermediate representations, as only the activations from the cut layer are shared with the server without the related functions to invert them back to raw data [3]. Furthermore, split NN is most useful in settings with a large number of clients, as split NN needs the least communication bandwidth, while federated learning does better with a relatively smaller number of clients. Additionally, R. Raskar et al. have researched different mechanisms for aggregating the outputs of the partial local networks, where they experimented on financial datasets [6].

Network "Configurations"

In another work [7], Raskar et al. propose different split learning configurations for practical settings in the health domain. In that paper, they refer to configuration in a broader sense than we use in this paper, as they mean the actual setup of a split learning algorithm which is not necessarily intended for vertically partitioned data. Besides their configuration for vertically partitioned data - which we employ in this paper - they also show configurations for different settings, such as the situation where the clients do not share any labels with the server, or the situation where pooled data from multiple clients goes to multiple servers, where each server solves a different supervised learning task (using different labels). In this paper however, we specifically research different split learning configurations for vertically partitioned financial data, although it is not limited to financial data at all; our observations on the cut layer and vertical partitioning can be generalized to other domains as well. These hyperparameters have not - to our knowledge - been researched yet, though they - if chosen poorly - can have detrimental effects on the split model's performance as we will show in this paper.

An Intuition

At a Data Council conference, P. Vepakomma answered an audience member's question regarding how to choose the optimal way of splitting the neural network when implementing splitNN. In Vepakomma's words "a split towards the end is better, but the problem is you're giving away in terms of the computation because there are more parameters in the ending layers" ... "if you are trying to reduce this [bandwidth/memory efficiency] loss, you can split at the second or third layer, which means the client has to compute very little" [8]. In general, this gave us a sufficient sense of intuition when contemplating where to split the network, but actual research to support this was not yet conducted.

Input Partitioning

Furthermore, how to optimally distribute the features among partitions for split learning has not been researched, though we did find material on the more general task of input partitioning based on correlation for neural networks [9]. This paper proposes that partitioning the input space based on the correlations between input features can improve the neural network's performance. In the paper, Guo et al. suggest that there can be positive and negative interactions between attributes, which is why some groupings of input features can lead to poor results; if attributes with a positive effect on each other are trained, good performance might be attained. They then proceed to link these attribute interactions to the correlation coefficients and propose an algorithm which partitions the input space optimally to promote positive interactions and remove negative interactions, leading to better performance. Though - since that algorithm automatically partitions in any amount of partitions instead of 2 - their proposed algorithm for partitioning the input has not been employed in our work, although the work did give us a direction to look in when finding the optimal way of distributing features, which was looking at the inter-feature correlations and trying to relate them to the split model's performance.

Experiments

Datasets

The split NN method is implemented on several different financial datasets. Each of these are used for a binary classification problem.

- *Synthetic Financial Datasets For Fraud Detection* (Kaggle, 2017 [10]) is a dataset synthetically created to fill the need of publicly available financial datasets, which are scarce because of their sensitive nature. The data is generated using their proposed PaySim generator, which uses one month of real financial logs from a mobile money

_	synthetic	healthcare	insurance
#samples	307511	5410	1000
#dim	11	28	43
#classes	2	2	2

Table 1: Datasets before preprocessing. #samples denotes the number of samples, #dim denotes the number of features, #class denotes the number of target classes.

service provider implemented in an African country to generate realistic looking financial data. To create fraudulent behaviour, PaySim implements fraudulent agents which aim to make profit by taking control of customer accounts and try to empty their wallet by transferring to another account and then cashing that money.

- *Healthcare Provider Fraud Detection Analysis* (Kaggle, 2019 [11]) is a dataset created for the task of predicting provider fraud; a form of organized crime which involves peers of providers, physicians and beneficiaries acting together to make fraud claims, which is one of the biggest problems facing Medicare, USA's national health insurance program assisting Americans aged 65 and older, or young people with disabilities. This act of intentional deception of the healthcare system - by for e.g. administering diagnosis codes which involve unnecessarily costly procedures and drugs is a driving factor behind the ever-increasing price of insurance premiums in the USA. Some types of healthcare fraud committed by providers include [12]:

  – Submitting multiple claims for the same service
  – Billing for a service visit or supplies the patient never received
  – Billing for a more expensive service than the patient actually received

  This particular dataset required more preprocessing than the others, since the data consists of three tables, which were also already divided into test and train. Only the train data was used, since the test data was not labelled.

  1. Inpatient data consists of data about the claims filed for patients admitted into the hospital, such as the date of admission and discharge and diagnosis code.
  2. Outpatient data consists of data about the claims filed for patients that are not admitted into the hospital.
  3. Beneficiary details data consists of general details on the person designated to receive the insurance benefits, such as location, gender, race and health condition.

  To utilize this data optimally and make it useful for a neural network, the data is preprocessed such that we end up with a table consisting of rows of providers, and the target variable, fraud or non-fraud. In order to aggregate the three tables into one, we e.g. sum the reimbursed insurance claims per provider, and count the number of patients/claims/physicians/procedures involved per provider. All the features are thus engineered and is only numerical, unlike the other datasets where one-hot encoded categorical data is included. Similarly, the admission/discharge dates were transformed into the number of days admitted to the hospital.

- *Insurance Claim* (Kaggle, 2019) is a small and seemingly synthetic dataset of insurance claims; though there is no information available on Kaggle on the data's source, certain features such as 'insured_hobbies' and 'insured_occupation' and their values definitely make this data seem synthetic. For the scope of this paper however, this is not problematic. In fact, having these types of features is quite coherent with this paper's aim of analyzing vertically split data; one entity holds data on the insurance claims - such as incident type/date/location/time etc. - whereas another entity holds identifying data on the insurance claimer.

After all preprocessing, which includes applying one-hot encoding to all categorical features, applying SMOTE (which will be discussed in the next paragraph), dropping features with more than 30% missing values, and in the case of the 'healthcare' dataset, dropping about half of the dataset due to lack of labels, we end up with the properties in table 2.

_	synthetic	healthcare	insurance
#samples	5966	9808	1506
#dim	10	28	1089
#classes	2	2	2

Table 2: Datasets' properties after preprocessing. #samples denotes the number of samples, #dim denotes the number of features (including one-hot encodings), #class denotes the number of target classes.

Class Imbalance

For each of these three tasks, there is an inherent class imbalance, as fraud is much less common than non-fraud, and it needs to be dealt with. Balancing this data is done using SMOTE [13], which imputes new instances of the underrepresented class having plausible features interpolated from the existing instances. This has proven to be very useful in improving all models' performance. This does increase the number of samples, so table 1 is not precise after preprocessing. Since categories need to be one-hot encoded to be used, and this data is binary, regular SMOTE would impute new non-binary data for these one-hot encoded features. For these cases, SMOTE-NC is used, which can handle categorical data.

Evaluation Metric

In each of these three tasks, there is an inherent cost imbalance to be considered in practical settings. Namely, wrongly accusing customers/healthcare providers/patients of committing fraud can be very harmful to a business' reputation and can lead to all sorts of harm, including expensive lawsuits (see Dutch childcare benefits scandal [14], which has destroyed many lives by wrongly accusing many parents of committing fraud when claiming childcare benefits). For our datasets we therefor look mostly at the F1 metric when comparing performances, which is the harmonic mean of the model's precision and recall.

Multiple Clients

For all datasets we vertically split the data over only two clients. Firstly, this is because with vertically distributed data there are - unlike in horizontally distributed data - usually not too many sources with useful data about a common user. Furthermore, we stick to two clients because the dimensionality of the used datasets is quite low, and limiting the code's complexity is also advisable for the aim of this project.

Standard Scaling

Additionally, all datasets are standard scaled before split NN is implemented, i.e. for each feature, the mean is substracted and it is scaled to unit variance. Not doing this has proven to be very harmful for the split NN model, even when the centralized version performs well without standard scaling.

Comparison to a Centralized Model

In Table 4 we compare the results of training a centralized model (M) with training several split models (M_1, M_2, M_3, ... M_{CL}) each distributed over two clients. Of all columns (C), each column constitutes a possible vertical split index, and each of the neural network's total number of layers (L) is a candidate cut layer. The hyperparameters of the centralized models were optimized using Optuna [15], an automatic hyperparameter optimization framework, particularly designed for machine learning. The optimal set of hyperparameters were picked based on their F1-performance, and they can be found for all datasets in Table 3. The same hyperparameters have been used when creating split models. All models are trained using the binary cross entropy loss criterion and batch sizes of 64. Merging the outputs of the split models has been implemented using element-wise concatenation. The models are trained for up to 100 epochs, with early stopping activated after the first 10 epochs to avoid overfitting. All datasets are divided into three subsets: a training set (60%) is used to train the network; a validation set (20%) is used to evaluate the quality of the neural network and to avoid overfitting during the training; finally, a test set (20%) is used to evaluate the resulting network.

As seen in table 4, results vary substantially between datasets; applying vertical partitioning on the Healthcare Provider dataset results in virtually on-par performance in every attempted configuration with little variance, while applying split learning on the other 2 datasets results in some poorly performing, as well as models which performance even exceeds it's centralized counter part.

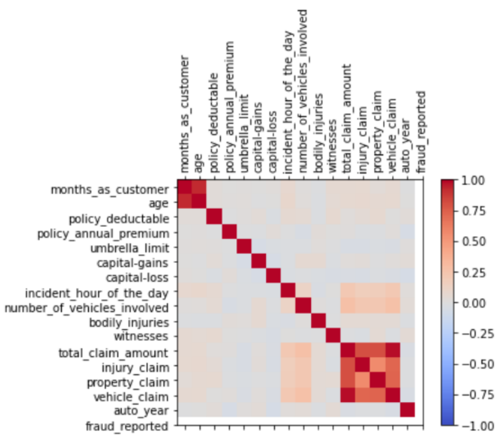
Dataset	N layers	N nodes L0	N nodes L1	N nodes L2	N nodes L3	Optim	LR
Synthetic Financial	4	111	79	87	118	RMSprop	0.0083
Insurance Claims	4	1592	934	633	118	Adam	0.0002481
Healthcare Provider	3	20	33	16	N/A	Adam	8.5601e-05

Table 3: The hyperparameters for all three centralized neural networks, optimized using Optuna [15]. 'Optim' denotes the optimizer used during training for minimizing the loss function. Both Adam and RMSprop are gradient-based optimization techniques. 'LR' denotes the learning rate used, and the other columns denote the architecture of the neural network (i.e. number of layers, and for each layer the number of nodes in that layer).

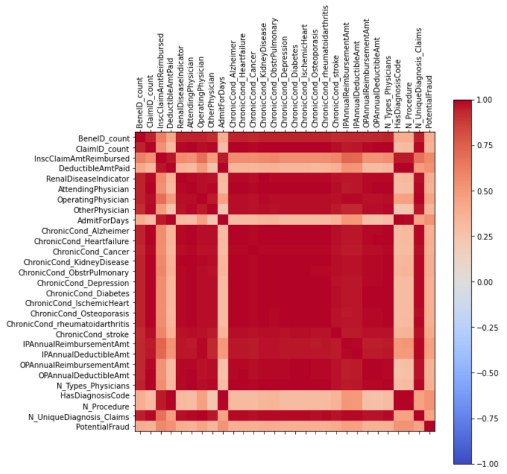
	Centralized Model	Split Model		
Dataset	F1	F1 (avg)	F1 (best)	F1 (worst)
Synthetic Financial	0.9733	0.9243	0.9842	0.6111
Insurance Claims	0.9460	0.9440	0.9938	0.7618
Healthcare Provider	0.8613	0.8632	0.8775	0.8439

Table 4: Comparison of the performance of a single centralized model with full data access and optimized hyper parameters vs the average performance of all $C * L$ possible split models with two vertical partitions.

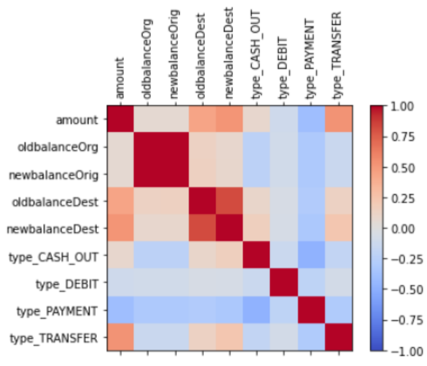
The results from the Healthcare Provider dataset (Table 4, Table 6, Figure 5) all stay in a small range of F1 performance, namely between 0.8439 - 0.8775, which is quite significantly stable, as the other datasets' models endured a lot more irregularities in performance for certain configurations.



(a) Insurance Claims



(b) Healthcare Provider Fraud



(c) Synthetic Financial Dataset

Figure 3: Correlation matrices for the three datasets. These plots show the correlation coefficients between features, indicated using color. Note that for 3a not all features have been included.

Comparison of Architectures

The following plots have been made in order to visually compare architectures and potentially see what the culprits of drops and peaks in performance are. Note that in this paper we count the cut layer k value starting from 0 at the output layer.

Synthetic Fraud Dataset

cut layer k	F1-score (avg)	Standard Deviation
0	0.9199061414609985	0.08038361742407205
1	0.9557223820206697	0.022623391431598864
2	0.9234423412437844	0.055248367306684296
3	0.9118882846845172	0.09589656835462293

Table 5: Average and SD of 3 reruns of generating all attempted split models' F1-score, grouped per cut layer k for the Synthetic Fraud Dataset.

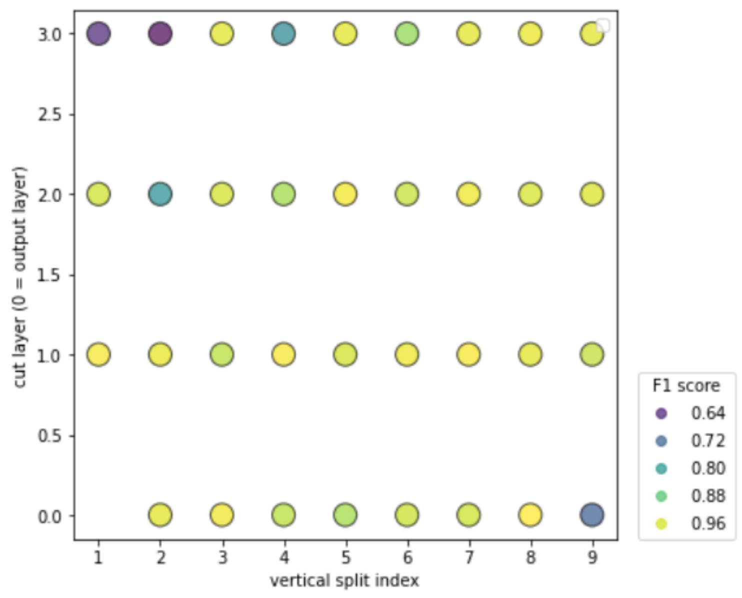
The generated results for the Synthetic Fraud Dataset were the least coherent to our expectation, since the lowest F1-performances were achieved with a high k value (i.e. cut layer close to the input layer). Especially the outliers are interesting, e.g. the models with $k = 3$ and vertically splitting at an index below 3 generate the lowest performance. Table 5 also contradicts our hypothesis; on average, k=3 and k=0 render the worst performing models. What can be concluded is thus that a generalization such as "performance increases with the cut layer chosen closer to the input", which was our initial hypothesis, cannot be made. Table 5 also shows that employing a cut layer of k=3 or k=0 result in highest standard deviations, i.e. the resulting model's performance is most unpredictable for these k-values. Furthermore, splitting at vertical index 2 most consistently results in sub-par performance, but it gets better as the cut layer k decreases.

Healthcare Provider Fraud

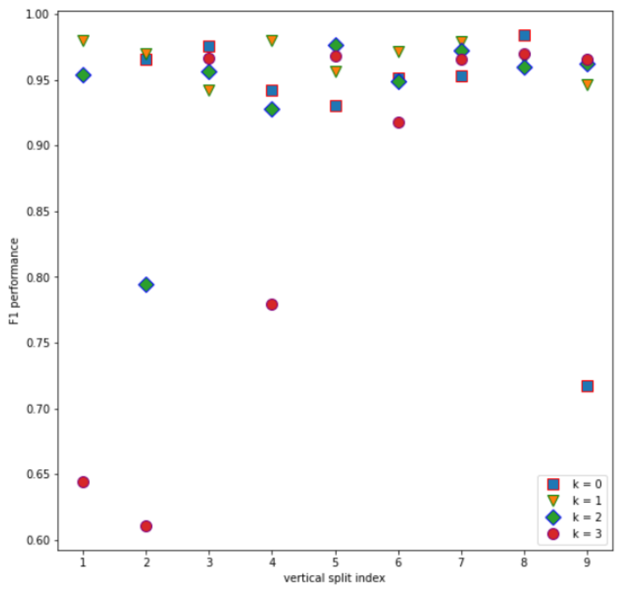
As mentioned before, the split models for the Healthcare Provider Fraud dataset all consistently performed in a small range of F1 = 0.846 - 0.876, which is noteworthy, as the other models endured a lot more turbulence when attempting different configurations. As the differences for this dataset are so small, no significant observations or conclusions can be made regarding the effects of different vertical split indices or different cut layers. Furthermore, splitting the data vertically did not have significant negative impact on the model's F1 performance.

Insurance Claims Fraud

The Insurance Claims Fraud Dataset produced some more coherent looking plots when comparing different configurations. Since this dataset's dimensionality expands so much



(a) F1 score denoted using colors.



(b) F1 score on y-axis and different shapes as k-values.

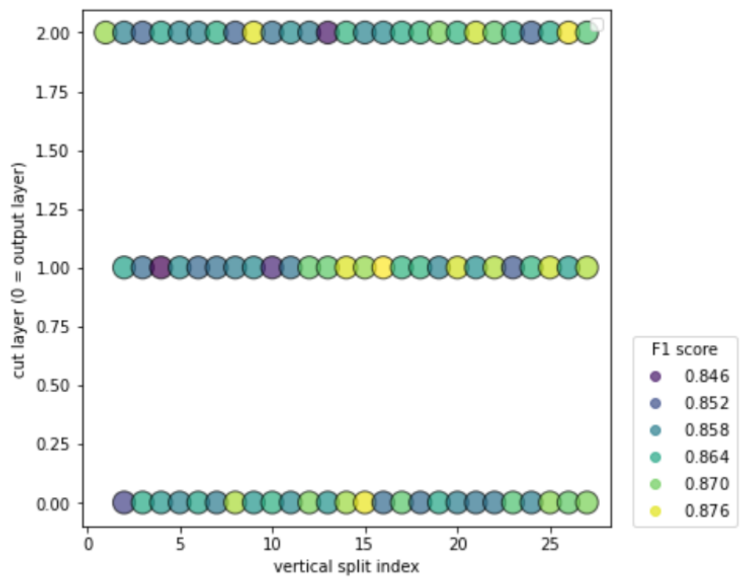
Figure 4: Synthetic Fraud Dataset; 2 visualisations of F1 performance scatter plots for different vertical split learning configurations. The x-axis is in both figures the index of the column at which the data has been partitioned. The y-axis is the cut layer in the left figure, and the F1 performance in the right figure.

after applying one-hot encoding, we end up with more fine-grained plots when attempting a vertical split at each of these indices.

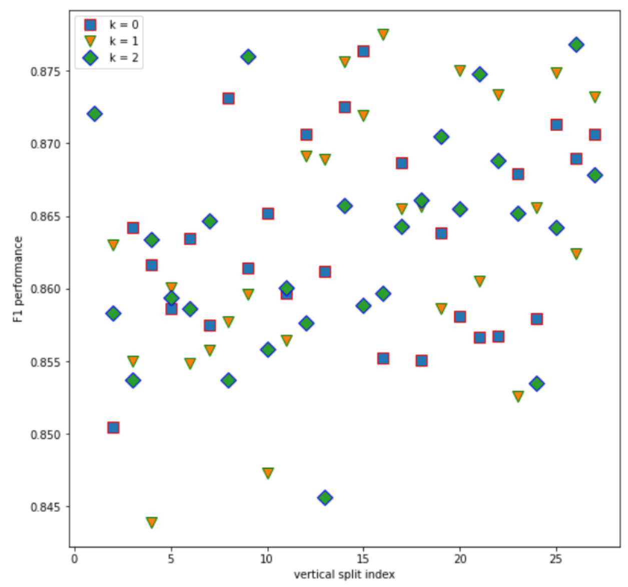
Especially Figure 6a contains some useful information regarding the effect of different configurations on the F1-performance. As you can see, the closer we choose the cut layer to the output layer (i.e. the more layers are trained locally among clients), the lower the F1-performance is on average. Thus for this dataset our hypothesis is confirmed. This assumption is also confirmed when looking at the average F1-scores for each cut layer k; the average F1-score increases as we choose the cut layer closer to the start of the neural network (see table 7). The standard deviations are also noteworthy; the higher the k-value, the lower the standard deviation of the performances, i.e. for this dataset we can conclude that joining

Table 6: The average and SD of all attempted split models' F1-score grouped per chosen cut layer k for the Healthcare Fraud Dataset.

cut layer k	F1-score (avg)	Standard Deviation
0	0.8633481687490474	0.006725433375777001
1	0.8632309244422489	0.00907346426445615
2	0.8629967575361028	0.00750107686184227



(a) F1 score denoted using colors.



(b) F1 score on y-axis and different shapes as k-values.

Figure 5: Healthcare Provider Fraud; 2 visualisations of F1 performance scatter plots for different vertical split learning configurations. The x-axis is in both figures the index of the column at which the data has been partitioned. The y-axis is the cut layer in the left figure, and the F1 performance in the right figure.

partitions of a split NN model earlier more consistently results in well performing models; a low k-value results in less predictable performance. Furthermore, there is a clear correlation between certain indices and the F1 performance. Seemingly, choosing to split within the vertical index range 700 - 1000 results in a drop in performance, especially at k=0, as well as a small spot around 130 - 140. The majority of worst performing instances seems to employ k=0, which is what we expected. Moreover, this model's behaviour regarding different k-values is exactly opposite to the Synthetic Fraud split model's behaviour when choosing a value for k; for this dataset, average performance increases with a decreasing k, as expected, whereas for the Synthetic Fraud dataset, we saw the the average performance decreasing

Table 7: The average and SD of all attempted split models' F1-score grouped per chosen cut layer k for the Insurance Claims Dataset

cut layer k	F1-score (avg)	Standard Deviation
0	0.9352719618190483	0.029627618753393935
1	0.9409539637829665	0.024154042419468492
2	0.9477507613255899	0.016818373832393964
3	0.9519580571909392	0.01394802391736949

when increasing k. An important distinction here is that this dataset has a lot more features resulting in about a 100 times as many split models; the results of this dataset thus say more about the general effect of choosing different k-values, whilst the counter intuitive results of the Synthetic Fraud dataset have a higher chance of being due to randomness.

Comparison of Feature Distributions

Lastly, the effects of choosing different features left and right on the F1-performance have also been examined for both the Synthetic Fraud dataset and the Insurance Claims dataset. All possible ways of splitting the feature set in two partitions have therefor been generated and their split learning models are compared. To limit computing time, the maximum amount of features to be considered was arbitrarily chosen to be 9, because it meant all features could be used for the Synthetic Fraud dataset. To be precise, we are generating a split learning model for every possible way to split a set of n features in two disjoint sets of r_1 and r_2 features respectively, where $r_1 + r_2 = n$, and then compare these models in a scatterplot, where the x-axis is an aggregate function which summarizes the feature distribution. The formula for the number of unordered partitions is

$$\binom{n}{r_1, n - r_1} = \frac{n!}{r_1!(n - r_1)!}$$

Thus for splitting 9 features in every possible way we generate

$$\frac{9!}{8!1!} + \frac{9!}{7!2!} + \frac{9!}{6!3!} + \frac{9!}{5!4!} = 255$$

feature distributions. For every feature distribution we generate a split model for every possible cut layer, i.e. 255×4 split models are generated for both datasets, as both neural networks constitute of 4 layers.

In order to evaluate the merit of a certain feature subset, a metric must be defined to calculate how good a feature selection is. Fortunately, feature selection for machine learning is a research field on its own, and different metrics and theories have been defined related to our task. For this experiment, we assume that the correlation between features in each subset is relevant to the performance of the split model, and therefor we test our hypothesis that the performance of the 255 generated split models is related to the correlations between the features. In this work we employ the Pearson correlation coefficients, but we can opt for others too. The Pearson correlation is only informative when applied to linear relationships; it may be non-informative or in some cases even misleading when applied to non-linear related variables. One evaluation metric we use to assess how good a feature set is for the model's performance, is the correlation feature selection (CFS) measure, which evaluates subsets of features based on a simple hypothesis: "Good feature subsets contain features highly correlated with the classification, yet uncorrelated to each other" [16] .

$$s_k = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}}$$

Here, $\overline{r_{cf}r_{cf}}$ is the average value of all feature-classification correlations, k is the number of features, and $\overline{r_{ff}r_{ff}}$ is the average value of all feature-feature correlations. For this experiment, we add the CFS value of both partitions (left and right) to summarize any instance's feature distribution.

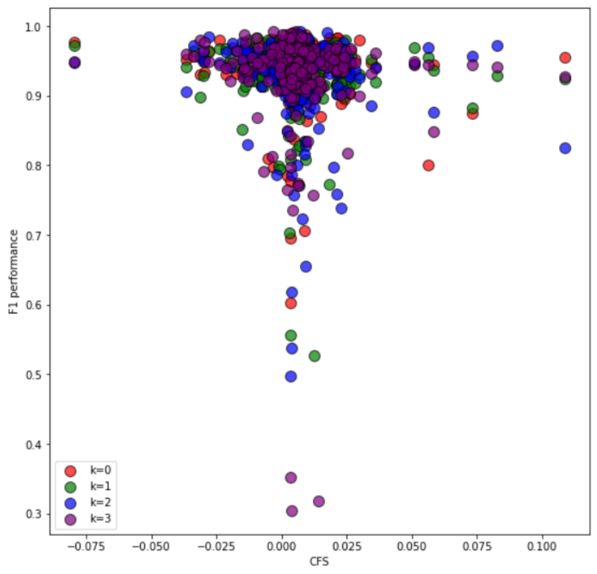
Experimental Results for Generating Feature Distributions

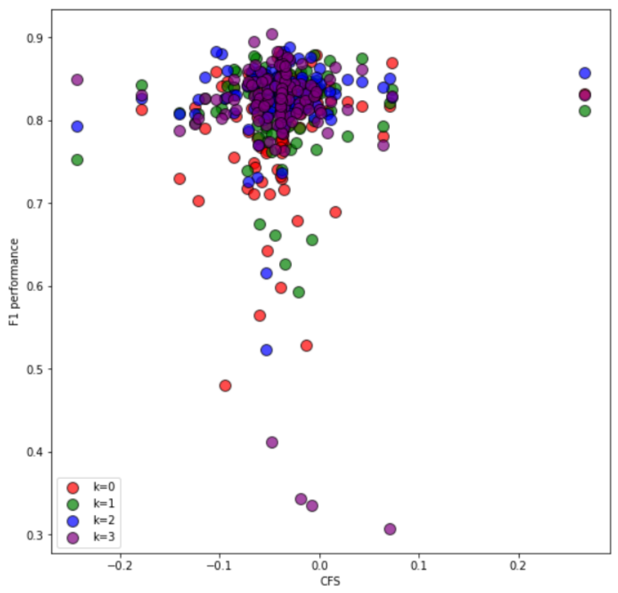
For the Synthetic Fraud dataset we utilize all features, and for the Insurance Claims dataset a subset of the most important features (determined using SelectKBest algorithm [17]) was used. This experiment is designed to test the hypothesis that there is a relation between the performance of the split model, and the (Pearson) correlations between all features of a partition.

Some interesting results were generated in these experiments by plotting the F1-performance against the sum of both partitions' CFS value, which can be seen in Figure 7a. First and foremost, both plots show the vast majority of cases around the CFS-value 0. Secondly, changing the distribution of features among partitions can clearly have significant impact on the model's performance, with the worst performing model achieving an F1-score of 0.3049. Another noteworthy result is that for both datasets, the worst performing split models all had k=3, yet the best performing cases also employed k=3. In the left plot we also see that - looking at the instances in ascending order of performance - the k-values decrease from k=3 to k=0, after which it becomes a lot more random.

Moreover, an important deduction which can be made from these plots is that the lower a split model's performance is, the higher the chance that it's CFS value is 0. From the plots we can also deduce that the further the partitions' CFS value is from 0, the higher chance that the model will not perform poorly. However, this implication does not go in the opposite direction, as there are plenty of cases where the CFS measure is around 0, yet the model performs well. A negative CFS value is also not necessarily indicative of a poorly performing split model. The validity of the relation we proposed between the split model's performance and the feature distribution is however not yet completely clear; the apparent relation we have visualized can also simply be a result of the fact that the vast majority of instances' CFS value is around 0, *including* the cases where the F1-performance is poor.

(a) Synthetic Fraud Dataset: F1 score on the y-axis and the CFS value on the x-axis.





(b) Insurance Claims Dataset: F1 score on the y-axis and the CFS value on the x-axis.

Figure 7: Effect of choosing different feature distributions, depicted using the relation between(summed) CFS and F1 for 2 datasets.

Discussion

The aim of our research was to compare and evaluate different configurations of a split NN model, with our research question being; *how do different vertically partitioned split NN configurations affect the model's performance?* In this section, we will discuss and interpret our results with the aim of answering this research question.

Cut layer k

Our first sub task was finding out how the cut layer hyperparameter affects a split NN model's performance. The most important conclusion here, was that a generalization such as "performance increases with the cut layer chosen closer to the input", which was our initial hypothesis, cannot be made; not only is there a lot more apparent randomness involved, we have shown cases (Table 5) where a model's worst performing models employed k=n, which is exactly opposite to our expectation. When rerunning the code responsible for generating all possible configurations for the Synthetic Fraud dataset, there seemed to be no consistency in what vertical indices are worst for splitting and what cut layer results in the best performance. Due to the small amount of features for this dataset, the code iterates a lot less than for the Insurance Claims dataset, which did clearly support our hypothesis. Another important observation with the Insurance Claims dataset, was that the higher we choose k to be, the more predictable the resulting model's performance is, i.e. the lower the standard deviation is. Bringing the partitions together in an early layer is thus more reliable performance-wise. For any given split NN model we suspect that if the features are predominantly uncorrelated, the cut layer behaves similarly to the Insurance Claims dataset; the higher we choose the cut layer value, the better and more robust the split NN model's performance. The Synthetic Fraud dataset, however, produced the least reliable performances when k=0 or k=n was employed, and the consistently best performing models when k=1 was employed, which we suspect being due to the features being a lot more correlated.

Vertical Split

Secondly, we wanted to find out how varying the feature distribution affects the predictive performance. For the vertical split research, we did not have a clear expectation on what would happen to a model's performance when changing the vertical splitting location. To our surprise, the Insurance Claims dataset generated very coherent plots with clear index ranges showing where splitting is generally harmful to the model (Figure 6). This led us to believe that, features surrounding those ranges need to be combined in the model for optimal performance; when separated, the model performs sub-par. The fact that the models' performance increasingly drops and the standard deviation increases when keeping the features separated for longer (table 7) adds more reason to believe this hypothesis. For this dataset, the worst performing models employed k=0, which is coherent with our hypothesis, because it means that keeping features that would rather be combined separate for longer results in worse performance. The Synthetic Fraud dataset, on the other hand, shows no clear relation between the vertical split index and the model's performance whatsoever. Our initial theory was that some features counteract each other when present in a partition together, explaining why a low cut layer value is better than a high cut layer value for this dataset. However, after 3 reruns of generating all possible configurations for this dataset, it became apparent that there was no consistency in what vertical split indices are bad for splitting, which refuted that hypothesis. The only observation that is quite conclusive here, is that vertically splitting the data in approximately equal parts is generally good for performance.

Counteracting Features

We suspect that the Synthetic Fraud experiments which refuted our hypothesis did so because some features can counteract each other, since experimenting with distributing the same features in different ways could apparently lead to huge drops in performance (see Figure7). We suggest that whether features are counteracting or not could be somehow related to the correlation coefficients of a feature set, as we saw an important distinction between the dataset with counteracting features and the one without was that the former had a highly varying correlation coefficient distribution (i.e. both negative and positive correlations), whilst the latter had a lot of uncorrelated features. From the results in Figure 7, we conclude that, the worse a split NN model performs, the higher the chance that the CFS value is 0. The CFS value depends on the average of feature-classification correlations, as well as the number of features in the set. Thus we suggest that if a split NN model performs poorly, changing the set of features in such a way that the average of feature-classification correlations strays further from 0, as well as equalizing the number of features per partition could help performance. This would however require further research to be made conclusive, as this observation in the data could have been by pure chance; most models simply have a CFS of around 0, thus there is a chance of the poorly performing models also having a CFS of 0.

Conclusion

In this paper, we compared and evaluated different split learning configurations for vertically partitioned financial data, and addressed specific challenges which arise in the architectural design of split neural networks. We have shown how some of the ways of partitioning data vertically can result in drastic drops in performance, and what might cause this.

On Generalizability

To summarize, whether to use a split model or not is very case and neural network specific, and relies heavily on how the input attributes are interacting with each other. What can be deduced from our experimental results, is first and foremost, that neural networks' feature interactions cannot be generalized; in some cases combining features and doing so early rather than later in the network is best for the performance, yet in other cases combining features early in the network (i.e. a high cut layer value) results in the worst performance. Our initial hypothesis "performance increases with the cut layer chosen closer to the input", can thus not be made. However, this does hold when we deal with features that are mostly uncorrelated. In that case, the closer we choose the cut layer to the input, the more consistent and optimal the resulting split model's performance is (Table 7).

Furthermore, varying the vertical split index can clearly result in significant drops in performance, when the features around that index need to be combined in order for the model to accurately predict. We saw this clearly when observing certain vertical index ranges where - if we vertically partition in that range - the model's performance clearly and consistently dropped (see Figure 6), leading us to believe that separating attributes in that range is harmful to that model. Additionally, we have observed that vertically splitting a feature set in approximately equal parts is - in general - better for the model's performance.

Correlations and Performance

We have thus shown that cases exist where split learning performs sub-par to a centralized model whilst there are also cases where the model performs on par or even better than its centralized counter part, and propose that it is related to the correlation coefficient distribution within that feature set. More specifically, we observe that all poorly performing split models use partitions with a correlation feature selection (CFS) value tending to 0. Exactly how these factors are related is yet to be defined, but we suggest to further research the effect of correlations and feature importance in and between partitions on the split model's performance. One hypothesis which could be tested in future research is that a feature set/partition is good for a neural network if the features in the set are as uncorrelated to each other as possible (to reduce redundancy), as well as highly correlated to the classification variable, and consisting of important variables, which can be determined using SHAP values (SHapley Additive exPlanations [18]); a unified approach to interpreting and explaining model predictions of any machine learning model, by measuring the importance of all features for the model's outcome).

On Social Impact

A concern from the author worth considering is that - though the novel advancements in split learning and algorithms alike are aimed towards fighting insecure data sharing and prevent misuse - these advancements do enable a lot more machine learning to be applied to sensitive data; even if it was never the creator's intent, we do enable reasoning about e.g. fraud prediction based on inferred characteristics, such as the person's background/heritage/race/sex. These biases must be considered at all times when designing systems like this, and the model's prediction should never be blindly followed. The prediction should - in the author's opinion - only be interpreted as a flag for further manual investigation. For the Insurance Claims experiments we did, we included a lot of features which in a real world scenario would be a lot more prone to bias, and we did in fact see the model's prediction being largely based off of the location of the incident, which is just one example of bias and overfitting. We therefor recommend to use the feature importance calculation using SHAPley values, to investigate your model's prediction and find out what features are mostly responsible for the prediction.

On Implementing Split NN

To conclude, we want to summarize our findings by explaining how we implement split NN to avoid poor performance, which is also visualized in Figure 8. First and foremost, it should be decided whether pooling data is even desirable at all for the task at hand. This can be answered by collecting a small ($n >= 1000$) subset of data about common instances from the data sources in question. This data can be used to check whether a neural network using the combined data outperforms neural networks using only data from one source or the other, which means a distributed deep learning network would be desirable. We found that standard scaling the input data can be very beneficial, as omitting it can result in a model that will not even train, even when the centralized version does train without scaling first. When choosing the cut layer, we recommend calculating the correlation matrix first and deciding whether it consists of i) mostly uncorrelated features, ii) mostly positively correlated features, or iii) highly varying between negatively and positively correlated features. In cases i) and ii), choosing a k of n or n - 1 is generally best (i.e. cut layer closest to the input). In case iii), choosing a middle layer as the cut layer is advisable. If the model still performs sub-par to the centralized version, the k-value can be decremented, or if the partitions are very unequal in size, they should be equalized by e.g. disregarding redundant features (i.e. where the correlation coefficient is approximately 1). If bandwidth or memory efficiency loss is too much in the resulting model, the k-value can be incremented as the client then has to compute less, but you are giving up some degree of privacy when splitting closer towards the input.

With this paper's findings, some barriers in implementing a vertically partitioned split NN model should be alleviated, hopefully helping the advancement towards privacy-aware distributed deep learning in health, finance and beyond.

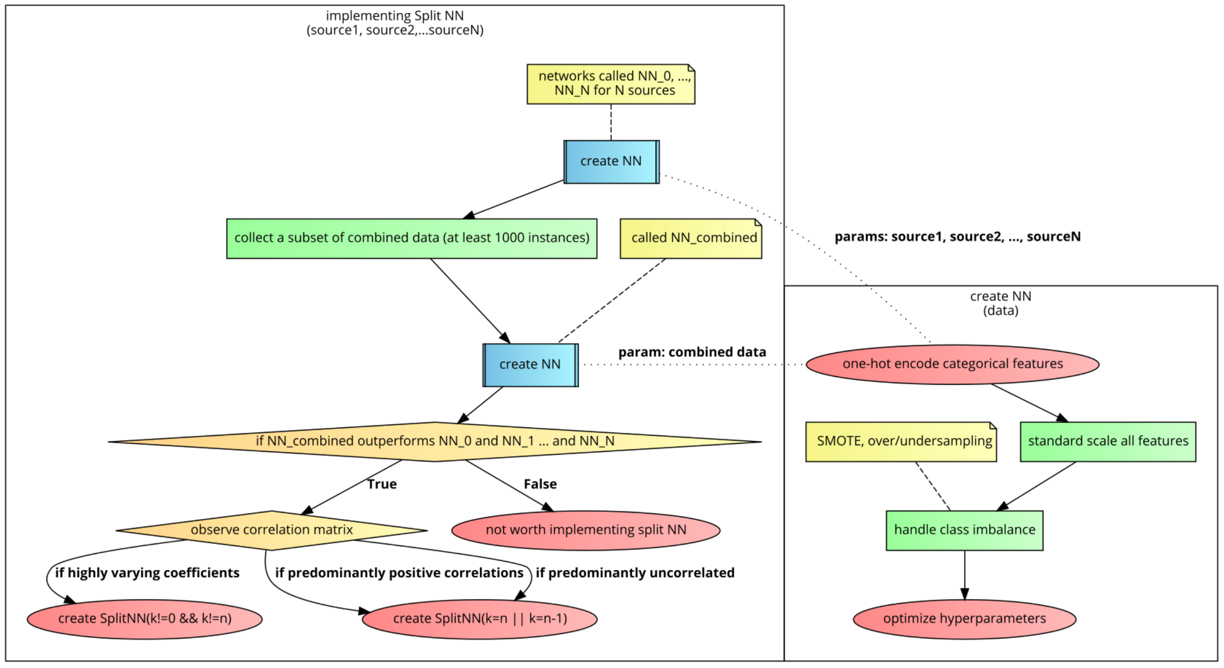
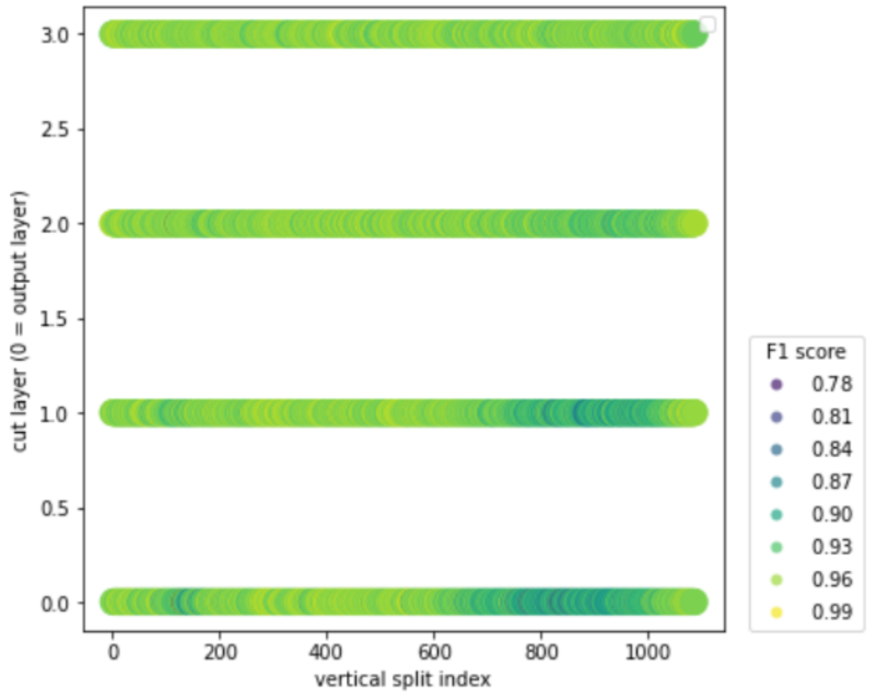
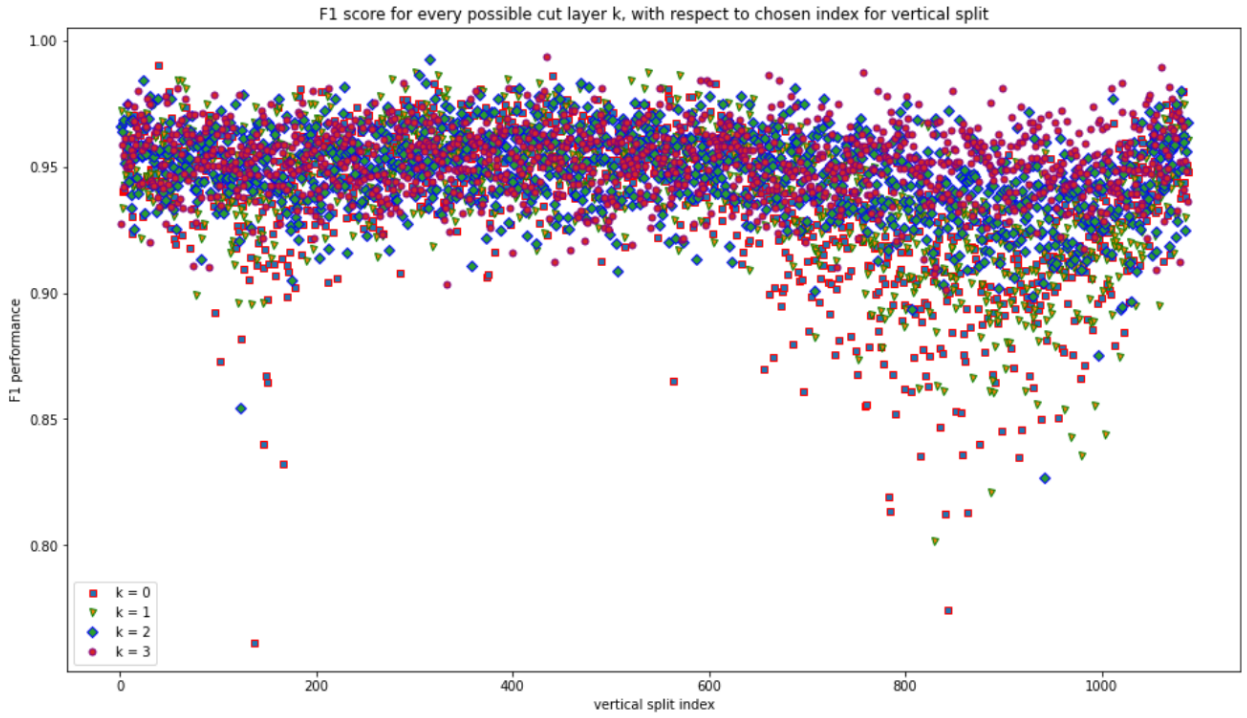

implementing Split NN
(source1, source2,...sourceN)
networks called NN_0, ..., NN_N for N sources
create NN
collect a subset of combined data (at least 1000 instances)
called NN_combined
params: source1, source2, ..., sourceN
create NN
param: combined data
if NN_combined outperforms NN_0 and NN_1 ... and NN_N
True
False
observe correlation matrix
not worth implementing split NN
if highly varying coefficients
if predominantly positive correlations
if predominantly uncorrelated
create SplitNN(k!=0 && k!=n)
create SplitNN(k=n || k=n-1)
create NN
(data)
one-hot encode categorical features
SMOTE, over/undersampling
standard scale all features
handle class imbalance
optimize hyperparameters

Figure 8: A flow chart visualizing how to decide whether split NN is useful for a task and which k-value to choose, based on our experimental findings. Yellow rectangles denote comments, diamonds are conditionals, blue rectangles are function calls, and the largest rectangles are the functions.



(a) F1 score denoted using colors.



(b) F1 score on y-axis and different shapes as k-values.

Figure 6: Insurance Claims Fraud; 2 visualisations of F1 performance scatter plots for different vertical split learning configurations. The x-axis is in both figures the index of the column at which the data has been partitioned. The y-axis is the cut layer in the top figure, and the F1 performance in the bottom figure.

Bibliography

[1] R. R. Otkrist Gupta, *Distributed learning of deep neural network over multiple agents* (2018).

[2] C.-D. J. B. A. L. C. . R.-M. M. Trevizan, B., *(2020). a comparative evaluation of aggregation methods for machine learning over vertically partitioned data.* (2020), `2020.113406`.

[3] R. R. O. G. Praneeth Vepakomma, Tristan Swedish and A. Dubey, *No peek: A survey of private distributed deep learning* (2018), `1812.03288v1`.

[4] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, in *NIPS Workshop on Private Multi-Party Machine Learning* (2016), URL `https://arxiv.org/abs/1610.05492`.

[5] L. Bottou, in *Online Learning and Neural Networks*, edited by D. Saad (Cambridge University Press, Cambridge, UK, 1998), revised, oct 2012, URL `http://leon.bottou.org/papers/bottou-98x`.

[6] I. Ceballos, V. Sharma, E. Mugica, A. Singh, A. Roman, P. Vepakomma, and R. Raskar, *Splitnn-driven vertical partitioning* (2020).

[7] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, *Split learning for health: Distributed deep learning without sharing raw patient data* (2018), `1812.00564`, URL `http://arxiv.org/abs/1812.00564`.

[8] P. Vepakomma, *A resource efficient distributed deep learning method without sensitive data sharing — mit*, URL `https://youtu.be/4PdBM76D5JY?t=2444`.

[9] S. Y. W. F. L. L. F. Z. Shujuan Guo, Sheng Uei Guan and J. H. Song (Journal of Clean Energy Technologies, 2013).

[10] A. E. E. A. Lopez-Rojas and S. Axelsson, *Paysim: A financial mobile money simulator for fraud detection*, The 28th European Modeling and Simulation Symposium-EMSS (2016).

[11] R. A. Gupta, *Healthcare provider fraud detection analysis* (2019), `https://www.kaggle.com/rohitrox/healthcare-provider-fraud-detection-analysis`.

[12] FBI, *Health care fraud*, website (2016), retrieved May 31, 2021 from `https://www.fbi.gov/scams-and-safety/common-scams-and-crimes/health-care-fraud`.

[13] N. V. C. et al., *Smote: Synthetic minority over-sampling technique* (2002).

[14] S. Amaro, *Dutch government resigns after childcare benefits scandal*, URL `https://www.cnbc.com/2021/01/15/dutch-government-resigns-after-childcare-benefits-sc`

[15] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework* (2019), `1907.10902`.

[16] M. Hall, *Correlation-based feature selection for machine learning* (2000).

[17] T. Desyani, A. Saifudin, and Y. Yulianti, IOP Conference Series: Materials Science and Engineering **879**, 012091 (2020).

[18] S. M. Lundberg and S.-I. Lee, *A Unified Approach to Interpreting Model Predictions* (Curran Associates, Inc., 2017).