Stockholms universitet

Personalization by Combination

The promises and problems of multi-armed bandit orchestrated federated learning

Fredrika Lundahl

Masteruppsats i matematisk statistik
Master Thesis in Mathematical Statistics

Masteruppsats 2021:14
Matematisk statistik
Juni 2021

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Master Thesis **2021:14**
http://www.math.su.se

Personalization by Combination

The promises and problems of multi-armed bandit orchestrated federated learning

Fredrika Lundahl*

June 2021

Abstract

This thesis is written in collaboration with Ericsson AI Research and is investigating the statistical aspects of a novel, yet unpublished, method to conduct personalized federated learning. The method is the result of a search for a communication efficient personalization algorithm which is also good at handling poisonous workers.

The proposed model originates from the Federated Averaging algorithm but replaces the averaging over the models of all (or most) workers by the averaging over the models from a selected combination of workers only. The combination is selected by a multi-armed bandit inspired decision rule, which learns the appropriate set of workers to combine in order to optimize the performance on a given target worker. Consequently, the decision rule also learns which workers to avoid.

The model is the most advantageous when the number of workers is small and each worker has a small dataset, situations where other methods struggle. However, having a reinforcement learning model inside a federated learning model causes some challenges, mainly relating to variance caused by the federated learning model. Some ways to combat these challenges are proposed.

By incorporating cluster analysis in future works, we might be able to give the model wider applications as it then should be able to personalize successfully even when the number of workers is big and each worker contains more data.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: ffredrikaa@gmail.com. Supervisor: Chun-Biu Li.

Acknowledgements

I wish to thank my university supervisor professor Chun-Biu Li for his help, guidance and endless support during this project. The discussions, inputs and comments have been much appreciated. His devotion to his students is endless and for that I am very thankful.

I would also like to express my gratitude towards my external supervisor Hannes Larsson at Ericsson AI research for his support, encouragements and valuable inputs on my adjustments of the model originally proposed by Ericsson. Also, last but certainly not least, for writing and providing the thousands of rows of code needed to run the experiments in this thesis.

Contents

1 Glossary

Federated Learning

Federated learning: Situation where the global (main/general) model is trained by aggregating the parameters of models trained on local similar, but not necessarily identically distributed, data sets. Data cannot be pooled nor accessed directly because of privacy and secrecy constraints.

Personalized federated learning: Federated learning system aiming to improve the local models instead of finding a good common general global model. The training of a general global model however often plays an important part of personalized federated learning.

Worker: A local entity containing and training a model on its (private) data set. It also communicates with the server holding the global model. It is sometimes called client in the literature. The worker can be a device such as a mobile phone. To simplify the terminology we often simply say worker instead of addressing an object contained in the worker. For example, we may say "performance on the worker" when meaning the performance on the data held by the worker, or "combine workers" when meaning combining the models trained by the workers, etc.

Cross-silo federated learning: federated learning where the workers are fewer and larger, such as entire organizations.

Poisonous worker: A worker with faulty (poisonous) data, created either by mistake or with purpose.

Aggregate models: Combining different models by, for example, taking the average of the model parameters as performed in Federated Averaging and this thesis. Here this will be equivalent to *combining workers*.

Round: One iteration of a federated learning algorithm, starting when the server holding the global model is sending out the model parameters to the selected workers and ending when it is receiving the updated model.

Communication: Exchange of information, such as model parameters or performance details, between the server and the workers. It is often limited and considered expensive.

Federated Averaging (FedAvg): Standard federated learning algorithm training the model with stochastic gradient descent (or a version of) locally on each worker before aggregating the trained models by averaging their model parameters.

Reinforcement Learning

Reinforcement learning: A machine learning paradigm differing from supervised- and unsupervised learning. The aim is to learn what actions to take in different situations in order to reach a well defined goal. The system learns by executing different actions and making inference based on the received feedback.

Action: A possible choice of action. To learn which action to take is to learn which choice to make. In this thesis an action corresponds to (selecting) a specific combination of workers.

Reward: The feedback from executing an action. When a reward is given, one receives a sample from the reward distribution of the action. Rewards are defined in accordance with our goal. Finding the action with the best expected reward in an efficient manner, meaning by taking as few samples as possible, is often necessary in order to reach our goal.

Decision rule: Policy for which action to execute given the situation and the collected information about the rewards. The decision rule is our strategy to find the action with the best expected reward as efficiently as possible. It may be stochastic and choose different actions with different probabilities. Several decision rules exist. In reinforcement learning literature the decision rule is called the agent.

Exploration: To explore is to execute actions estimated to be suboptimal in order to get more samples from the reward distributions of different actions. Exploration improves our estimates of the expected rewards and reduces uncertainly.

Exploitation: Opposite of exploration. To exploit is to trust the estimates and select the action with the best estimated expected reward as the next action.

Non-stationary rewards: If the rewards are considered to be non-stationary this means that the reward distributions are considered to change over time. Time is in this thesis the number of rounds of the federated learning algorithm. It is important to know whether we deal with stationary or non-stationary rewards to be able to correctly estimate the expected reward of the next action.

Multi-armed bandit: A simple and special case of reinforcement learning where we have only one state, meaning that we are repeatedly faced with the same situation. The name is an analogy for a casino slot machine (one-armed bandit) which instead of having one arm to pull has several. The probability of winning is different for each arm and the gambler wishes to figure out which arm gives the most wins.

Neural Networks

Neural network: State-of-the-art supervised learning technique to detect complex patterns in data. It consists of several layers of neurons connected together. The training of a Neural Network adjusts the strength (weights and biases) of the connections. Training is typically done with stochastic gradient descent as the network often has a lot (sometimes millions) of parameters.

Stochastic Gradient Descent (SGD): An efficient (compared to other techniques) method of training a Neural Network. Often when mentioning stochastic gradient descent one means mini-batch stochastic gradient descent, which will also be the case in this thesis. It implements Gradient Descent on a loss function *sequentially* on subsets (batches) of the data to ease up computations. Computing the gradient for one random subset of the data at a time makes it stochastic. Several extensions exist to improve its efficiency and precision.

Epoch: A full training iteration of a neural network. During one epoch the entire training data set goes through the network once, we can think of it as a training cycle. Predictions are made on the training data and the model parameters are updated in a manner which reduces the loss. The number of finished epochs gives the amount of training conducted on a network. Therefore too few epochs results in an underfitted model and too many may leads to an overfitted model if the performance does not converge.

2 Introduction

Federated learning is a machine learning method making it possible to conduct machine learning when having data distributed heterogeneously across different locations and being unable to collect it into a joint pooled data set. Typically we cannot collect the data because of privacy and secrecy reasons. Because of the restricted access we cannot inspect the data, but we generally have a good enough idea of its structure and what it might look like to assign a model. Federated learning utilizes the fact that while we cannot access the data, we can access the parameters of models trained to fit the data.

The rather standard Federated Averaging algorithm works according to the following: a global model is distributed to the workers (entities), on each worker it is then trained on the local data set. We get the new updated global model by aggregating the updated model parameters from the local models on each worker by taking the mean (McMahan *et al.*, 2017).

In the standard application of federated learning one wants to find a global model with *one* set of parameter values which is a compromise of all local models. Unfortunately, the one-size-fits-all approach often does not actually fit all and the fit of the global model can be quite bad on a local data set, especially in

federated learning where we deal with data which is of personal nature and therefore often heterogeneous. This problem gives rise to personalized federated learning, where each individual data set ends up with an adapted global model suiting the local data better. By doing so we wish to get the best of both worlds: insight from other related data sets and adaptation to the local data itself.

Previous works about personalized federated learning include, for example, adding personalization layers which are unique for each local model (e.g. Arivazhagan *et al.*, 2019). These are tuned during the training of a global model. Another simple approach is to train a global model which is later fine-tuned on each local data set (e.g. Fallah *et al.*, 2020). These are all quite expensive in communication, as they are based on Federated Averaging, and can be harmed by poisonous workers as all workers affect each others local models via their influence on the global model.

We propose a different way of conducting personalization by using a reinforcement learning approach and ideas from multi-armed bandits to find out which local models one should aggregate to improve the performance on a specific target data set. In that way we tune the global model to fit the data on a specific worker. Similarities exists to the method proposed by Wang et al (2020) where reinforcement learning is implemented in the form of deep Q-learning to learn which models that should be aggregated in every round. However, the aim in these is to reduce the communication of Federated Averaging but not to personalize.

A method proposed by Sattler *et al.* (2020) also aims to aggregate similar models and finds similarity by using clustering of the model parameters. Our approach does not include cluster analysis in its current state. Although there is great promise in combining the considered approach with clustering, that is beyond the scope of this project. In this thesis we try to find the workers complementing each other by the strategy of trial and error. This thesis is mainly about how to conduct and evaluate the trials, which problems we face and how we can ease them. If we can create a model finding the optimal combination of workers this will also mean that we find a model which avoids poisonous workers. Therefore we will not focus on the model's ability to avoid poisonous workers, but its ability to find the optimal combination.

We call the machine, which is supposed to learn which combination is favourable, the decision rule. The decision rule selects the workers to be aggregated each round according to some rule and is supposed to learn which combination of workers that has the best effect on the global model when evaluated on the target worker. This means that the decision rule learns *during* the training of the general global model. This approach can be categorized as online learning as the decision rule's training data (the effects of combinations) are not provided all at once but one by one as every round one combination is evaluated and its effect is provided to the decision rule.

Learning which combination is the best while training the model saves time as we learn what combination to use and tune the model parameters simultaneously. This is especially advantageous for complex models, such as neural networks, which are of particular interest in this thesis. However, as we train both the decision rule and the global model simultaneously, the performance of the global model is not only determined by the combination of workers but also is by the training itself. If we are in the early rounds, the performance of the global model will be quite bad no matter which combination we choose and given many training rounds it will be fairly good even if a suboptimal combination is selected. The parameters of the current global model are the initial conditions for each worker the next round. Therefore the training on the local models proceeds differently every round. This means that the effect we see of a combination, i.e., of the aggregated model of the selected combination of workers, will differ some between rounds and we consider the effect of a combination to be stochastic. One main problem is therefore how to best distinguish the effect of a specific combination, both from the training effect of the global model and from other combination effects.

The approach in this method is to utilize ideas from multi-armed bandits. In fact, we do not quite satisfy the conditions for multi-armed bandit models as we have an ever changing situation, due to the fact that we train a global model which changes with time and the decision rule is within that model. However, we try to standardize the effects by measuring improvement of performance such that each trial of a combination occurs in a similar setting. This makes out approach close enough to the multi-armed bandit situation.

The other main problem is that as we are collecting information about the effect of different combinations while training the global model, we can no longer train the decision rule after the global model has converged and finished training. We therefore need to find out how to get the most information out of a limited number of training rounds. A way to do this is to exploit the fact that the reinforcement learning system is in fact inside the federated learning system and this enables us to try out several combination per round of training.

We find that the approach is good when there is little data available in the target data set and when the number of workers is small. Our main application is cross-silo federated learning, where the number of workers is rather small. When there is little data, influence from other workers is the most advantageous. We want the number of workers to be small as the number of possible combinations grows exponentially with the number of workers and quickly become too many for us to be able to consider them all. As future work one should be able to extend the model to work in situations with a big number of workers by incorporating clustering analysis. Clustering analysis could limit the number of combinations to consider and thus enable us to evaluate each potentially good combination sufficiently. It might also enable us to aggregate workers with greater care by adapting the weights in the weighted average we implement when aggregating

models.

3 Federated Learning

The following section and subsection are based on McMahan *et al.* (2017).

Federated learning is in its simplest form a machine learning technique where we get one (shared) global model by aggregating local models. Because of privacy constraints, the actual training needs to take place on separate local devices with no knowledge of each other. A local device is called a worker (in some literature also called client). We can think of workers as containers of private data sets equipped with a model which can be trained to fit the local data. The model architecture across all workers is the same. The workers can communicate with the server. The workers send their updated models to the server where the models are aggregated into the new global model. The new global model is then sent to the workers where, again, training takes place.

All workers contain the same type of data, but we cannot assume the data to be distributed across the different workers in a balanced and homogeneous fashion. Consider for example workers containing health data for different individuals or data regarding business information for competing companies. This kind of data may vary greatly between different local instances and is also often of private nature and protected by great secrecy. Such data should never reach a cloud, even if made anonymous. Still, this data is potentially very valuable if one can find a way to use it. This is the kind of situation where federated learning is of great use, as its typical application is situations where data must not be taken out of its environment or even be inspected. Mostly this is because of privacy reasons but the reasons can also be computational.

In many federated learning situations one cannot rely on having stable access to the workers (e.g. the user holding the data set might not be connected to internet). Sending big sets of parameter values to and from workers can also be expensive and time demanding. The training of a good global model might therefore take very long time, making each round come with a (time) cost not in proportion to the number of computations made. Hence computational costs on the workers are often considered negligible, this is a stark contrast to most other machine learning situations, where communication is cheap and computation is expensive. Therefore, instead of minimizing computations we aim to minimize the need for communication with the server holding the global model by maximizing the amount of computation one can do locally on each worker. Communication includes all exchange of information, such as parameter values or information about the performance, between the server holding the global model and the workers.

The main research focus has been given to the situation where the model is in the form of a neural network, but the approach is not necessarily limited to that situation. However, the model needs to be a "black box" model if there are privacy constraints, as one can then not relate a parameter value to characteristics of the data sets.

In this thesis we will mainly focus on a special case called cross-silo federated learning. It is characterized by a rather small number of workers with non-i.i.d. data, for example a worker can be an organization (Kairouz, 2021).

3.1 Federated Averaging

A fairly simple algorithm that is still good and useful for federated learning is *Federated Averaging*, where optimization of the local models of each worker by stochastic gradient descent is followed by the local models being aggregating by model averaging. The approach has empirically showed to be robust against situations where data is unbalanced and distributed heterogeneously across the workers. Federated Averaging does not rely on, for this kind of data, the too simplified independent identical distributed (i.i.d.) assumption: that the training data is distributed across the workers uniformly. If one is confident about the data being IID there exists other algorithms like for example Federated Stochastic Gradient Descent (FedSGD) which is treating the data from each worker as identically distributed batches which are then used in SGD.

During one Federated Averaging training round, see Algorithm 1, the model parameters from the global model are sent out to each worker, where the model is then trained on the local data with SGD. Each worker then communicates the updated model parameters, the parameters from all workers are then aggregated by taking the weighted mean where the weight is proportional to the number of data points in the data set. The resulting model constitutes the new global model. This is illustrated for a linear regression model in Figure 1 and more generally in Figure 2. If we have a big number of workers we select some random proportion of the workers instead of all of them to speed up the algorithm.

If we train for more than one round the different models influence each other via their influence on the global model, which gives the initial conditions for every round of training. For the simple linear regression task in Figure 1 we know the form of the optimal model and more training rounds should not be needed. However, for more complex models with several local minima and no closed form expressions for the optimal values the initial conditions play huge roles in optimization algorithms.

Several rounds of Federated Averaging will often be needed to improve the global model as each round the workers get new, hopefully better, initial conditions

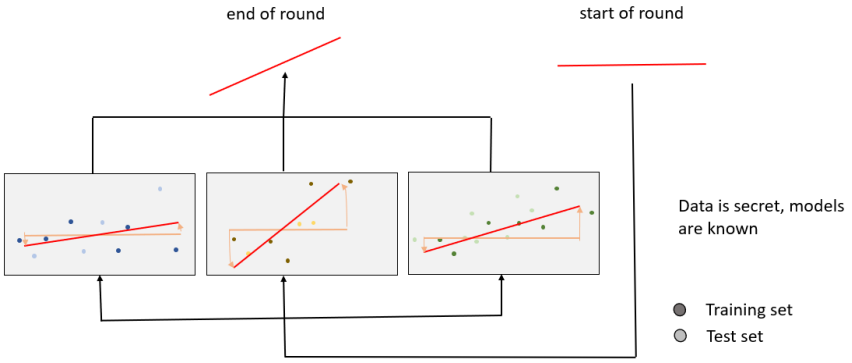

Figure 1: One round (iteration) of federated learning using Federated Averaging with a linear model and three workers. The current global model (a horizontal line) is sent out to each worker, where it is trained to fit the local data set. The models are then aggregated (by taking the weighted average) to a combined model, which constitutes the new global model. If we wish to continue training, this is the model that will be sent out to the workers in the next round. For a more general scheme see Figure 2.

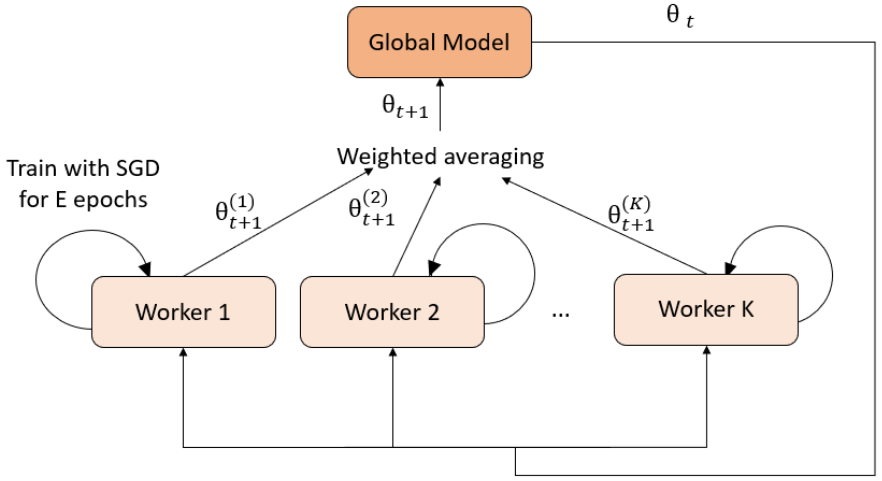

Figure 2: Federated learning with Federated Averaging. $\theta_t^{(k)}$ denotes the model parameters from worker k at round t. When averaging the model parameters from each worker are given weight in proportion to the size of the data set they are trained on. For an example with linear regression, see Figure 1.

Algorithm 1 Federated Averaging

Fix the parameters for SGD: the batch (subset) size B, the number of epochs (iterations of SGD) per round, E, the learning rate η and the loss function $l(\theta)$. Initialize the parameter values of the global model.
Iterate (for a predetermined number of rounds or until a stopping criteria is met):

1. Distribute the global model parameters $\boldsymbol{\theta}_t$ to all workers.

2. Perform SGD on the model on each worker. For worker k this means that we split the data into B batches and for each batch we update the model parameters $\boldsymbol{\theta}_t$ according to

$$\boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}_t^{(k)} - \eta \nabla l(\boldsymbol{\theta}; b), \quad b \in B$$

   and repeat for E epochs.

   The updated model parameters $\boldsymbol{\theta}_t^{(k)}$ are then communicated to the server.

3. Let K be the number of workers and n_k be the the number of data points in worker k, $n = \sum_{k=1}^{K} n_k$.

   We then aggregate the models from the different workers by taking the weighted mean:

$$\boldsymbol{\theta}_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \boldsymbol{\theta}_{t+1}^{(k)}.$$

   $\boldsymbol{\theta}_{t+1}$ is the new matrix containing the global model parameters.

which might help us avoid getting stuck in a bad local minima and prevent the local models from overfitting due to too few data points.

4 Reinforcement Learning

This section and subsection are based on Sutton and Barto (2018) unless stated otherwise.

Reinforcement Learning is about learning how to manoeuvrer situations with different choices of actions in order to reach a goal.

We do this by collecting and evaluating experience, which is done by trying out actions and mapping the results to rewards. Using statistical terminology: we take samples from the reward distribution of each action and use them to make inference about the expected reward of each action. "We" are embodied in the decision rule, the machine to be trained. The decision rule adapts its estimates of the expected rewards of different actions after each reward sample. It executes actions and takes samples from the corresponding reward distributions according to a strategy aiming to maximize the total accumulated reward in the long run, which is synonymous with reaching our goal. To reach our goal we need to find the action with the highest expected reward while being careful not to execute an unnecessary amount of suboptimal actions. However, we have no way of knowing whether an action is suboptimal or not without taking samples, which is why we need a good decision rule finding this out as cost-efficiently as possible.

The machine (the decision rule) is supposed to learn which actions give good rewards and use the acquired knowledge to reinforce itself after every reward sample. This makes reinforcement learning the machine learning paradigm which is the most related to how humans learn and many important algorithms are inspired by biological learning systems. It can be seen as a paradigm of its own as reinforcement learning strongly differs from supervised and unsupervised learning. It mainly distinguishes by being solely focused on goal-directed learning through interaction. To interact is to take an action affecting the situation we wish to learn about and receive feedback. More specifically, the differences are the following:

In supervised learning the correct actions are provided to the machine via training data and the machine is then to learn a good way to generalize the patterns presented to it. This is not the case in reinforcement learning, where the machine is supposed to learn the correct actions itself through interactions with its environment: training data is found rather than given. Training data also does not instruct the machine about the correct action but rather helps the machine to evaluate actions in order to find the one with the best expected

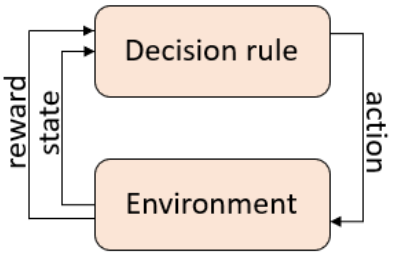

Figure 3: Basic setup for reinforcement learning. A decision rule takes the latest reward and the current state of the environment as inputs and updates its estimates. The decision rule then selects and returns the next action to execute (which reward distribution we should sample from). When the action is executed the environment changes its state and produces a reward which the decision rule uses as feedback.

value. For example, in a supervised learning problem we may wish to train a machine to be able to tell an image of a cat apart from an image of a dog. In order to do so we provide the machine with examples of images of cats and images of dogs which should be considered to be the truth. A reinforcement learning problem on the other hand may be to learn which route one should take to one's workplace in order to spend as little time as possible commuting. One then creates one's own training data by trying out different routes and mapping the time it took to a reward. As the best action should be the one with the highest reward we may let the reward be the negative travelling time. A single reward should not the considered to be the whole truth about the effect of an action but is merely a sample from a stochastic reward distribution. A travelling route may be the fastest one day but not the next due to traffic jam.

Unsupervised learning on the other hand focuses on finding hidden structures in the data. The purpose with reinforcement learning is not to understand the data but only to find a way to use it to reach our goal. For example, an unsupervised learning problem may be to identify different customer groups. The purpose of which is to gain additional knowledge, possibly in order to provide better service or display more appropriate advertisements. Reinforcement learning goals are more direct, such as finding the customer group which buys the most of a given product.

The most basic setup of a reinforcement learning system is with an environment and a decision rule interacting with each other. The environment consists of different states and contains all the possible actions and their reward distributions. The decision rule holds the strategy and the estimates of the expected rewards of each action. This is illustrated in Figure 3.

In models where the best action depends on the situation, we utilize the concept of states from stochastic processes. Markov decision processes are natural and important frameworks. Informally the state can be interpreted as the information available to the agent about the environment at the moment. For the commuting example it may be the day of the week or the time of the day. For a machine aiming to learn how to win a chess game the state is the chess pieces' positions on the board. It is possible to only have one state, which is the case for the standard multi-armed bandit situation which is used in this thesis.

To summarize: for a reinforcement learning model we need an environment, a decision rule and a reward signal. Optionally one can also have a model of the environment.

Decision rule: A mapping from the perceived state of environment and estimates of rewards to an action. The decision rule updates with every reward sample, which is given every time we try out an action. The decision rule can be stochastic and map to different actions with different probabilities.

Reward signal: Should be chosen in accordance with the goal of the problem. Every time we choose an action, a **reward** (a number) is sampled from the corresponding reward distribution and is sent to the decision rule as feedback. In that way, the decision rule can be updated and we learn to distinguish good actions from bad. Trying out actions is equivalent with sampling from their reward distributions. Rewards are often assumed to be stochastic. The objective is to maximize the total accumulated reward in the long run, for that we need to find the action with the best expected rewards as efficiently as possible.

Model of the environment: Describes how the environment behaves and allows for inference and planning before any experience is gained. For example, a model may subject to distributional assumptions. However, not all reinforcement learning systems include models of the environment but are explicitly trial-and-error.

Always taking the action which is estimated to maximize the expected reward is generally a bad decision rule, especially early in the training as the expected reward of each action is probably incorrectly estimated without seeing enough reward samples. However, trying out too many bad actions can be expensive and it does not lead to a maximal accumulated reward. We may also waste time, which might be precious, and other resources. Therefore the trade-off between exploration (taking samples from reward distributions of actions estimated to be suboptimal) and exploitation of what is known is central in reinforcement learning and the decision rules aim to balance the two. Exploration reduces uncertainty and exploitation utilizes maximum likelihood, where the likelihood function is the estimate of the expected reward given an action and the received reward samples.

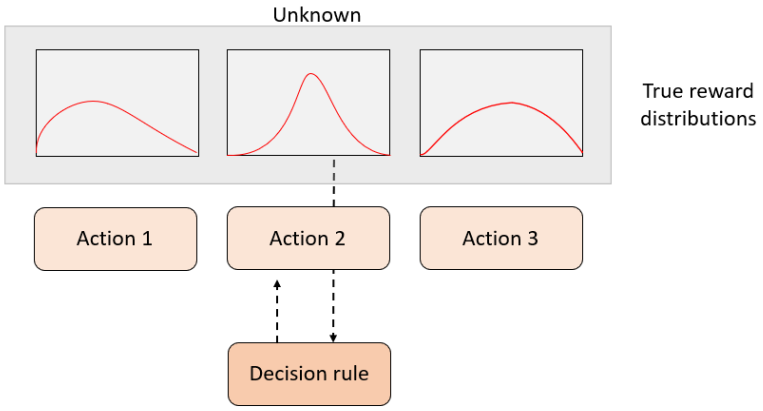

Figure 4: A simple multi-armed bandit situation with three different possible actions and the true corresponding reward distributions, which are unknown to the decision rule. Action 2 is selected by the decision rule and feedback is given by a sample from the reward distribution. The estimate of the expected reward of action 2 is updated.

4.1 Multi-Armed bandits

The classical multi-armed or k-armed bandit problem regards the situation where we are repeatedly faced with the same k choices of actions in the same setting, i.e., the environment only has one state. As the name suggests we may imagine ourselves standing in front of one k-armed bandit (or k one-armed bandits) or slot machine and we want to pull the arm that will give us the most money and avoid the ones prone to make us bankrupt. The probability of winning is different for each arm but unknown to us.

With each choice of action (arm) made by the decision rule we receive a numerical reward from the corresponding reward distribution, which we will assume is stochastic. In fact, choosing an action is equivalent to taking a sample from the reward distribution of the corresponding action. An illustration is showed in Figure 4. Our objective is to maximize the total reward over some long time period, which will require us finding the action with the biggest expected reward as effectively as possible.

Denote the random variable describing the action at time t by A_t and the corresponding reward by R_t. For an arbitrary action a we denote the expected reward q at time t by

$$q_t(a) := \mathbb{E}[R_t | A_t = a].$$

We denote the empirical estimation of $q_t(a)$ by $q_t^*(a)$ which we may obtain by taking the time average of the reward samples obtained up to time t, i.e.

$$q_t^*(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{N_{t-1}(a)} \tag{1}$$

The division with $N_{t-1}(a)$, the number of times we have taken action a up to time $t-1$ gives each observation the same impact. This might not always be good, as in many situations the rewards are *non-stationary*, meaning that the probability of observing a specific reward might be different at different times. With time we mean the number of executed iterations of the reinforcement learning algorithm, which is equal to the number of executed actions. If the rewards are non-stationary we should give more impact to the most recent samples, which should be more reliable and relevant than older samples when it comes to expecting the reward of the next action. To do that we note that we can rewrite the estimated expected reward as

$$q_{t+1}^*(a) = q_t^*(a) + \frac{\mathbb{1}_{A_t=a}}{N_t(a)}\big(R_t - q_t^*(a)\big) \tag{2}$$

since

$$\begin{aligned}
q_{t+1}^*(a) &= \frac{\sum_{i=1}^{t} R_i \cdot \mathbb{1}_{A_i=a}}{N_t(a)} \\
&= \frac{1}{N_t(a)}\left(R_t \mathbb{1}_{A_t=a} + \sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}\right) \\
&= \frac{1}{N_t(a)}\big(R_t \mathbb{1}_{A_t=a} + q_t^*(a) N_{t-1}(a)\big) \\
&= \frac{1}{N_t(a)}\big(R_t \mathbb{1}_{A_t=a} + q_t^*(a)\big[N_t(a) - \mathbb{1}_{A_t=a}\big]\big) \\
&= q_t^*(a) + \frac{\mathbb{1}_{A_t=a}}{N_t(a)}\big(R_t - q_t^*(a)\big) \quad \square
\end{aligned}$$

This makes what is happening in the updates more clear and we can see that to update the mean we do not need to recompute it but just update it with the weighted difference between the observed reward and the expected. This should be exploited in algorithms to save memory and time.

If we wish to give more weight to more recent samples, we can implement

exponential recency weighting. We then replace $1/N_t(a)$ with the constant $\alpha \in (0, 1]$, i.e.,

$$q^*_{t+1}(a) = q^*_t(a) + \mathbb{1}_{A_t=a}\alpha(R_t - q^*_t(a))$$

In the derivation of this we refrain to write out indicator to ease up the notation. By rewriting and expanding the above we get

$$\begin{aligned}
q^*_{t+1} &= q^*_t + \alpha(R_t - q^*_t) \\
&= \alpha R_t + (1-\alpha)q^*_t \\
&= \alpha R_t + (1-\alpha)(\alpha R_{t-1} + (1-\alpha)q^*_{t-1}) \\
&= \alpha R_t + (1-\alpha)\alpha R_{t-1} + (1-\alpha)^2 q^*_{t-1} \\
&= \alpha R_t + (1-\alpha)\alpha R_{t-1} + (1-\alpha)^2\alpha R_{t-2} + ... + (1-\alpha)^{t-1}\alpha R_1 + (1-\alpha)^t q^*_1 \\
&= (1-\alpha)^t q^*_1 + \sum_{i=1}^{t} \alpha(1-\alpha)^{t-i} R_i
\end{aligned}$$

and it is then clear why this weighted average is called an exponential recency average. It gives more impact to recent observations as we have an exponential decay in the reward coefficient. The weights still sum to one, the proof can be found in the appendix. The effect for some different values of α is showed in Figure 5. We note that the smaller the α the more weight is given to recent samples.

Choosing the action with the highest estimated value, i.e. $A_t = \text{argmax}_a\{q^*_t(a)\}$ is called a *greedy* action and choosing greedily corresponds to exploitation. This is the same as using maximum likelihood as a decision rule. However, it might be that the estimates of the expected rewards are bad and misleading due to too few samples from the different reward distributions. To be able to get a more reliable estimate it is therefore vital that we to some extent, especially early in the experiment, do exploration: choose actions estimated to be suboptimal. This is especially important in the beginning as we may otherwise get a bias which remains during a long time. One way to encourage extra exploration in the first rounds is to set high initial values. This will make the decision rule execute all actions during the first rounds as the non-executed actions will have the highest estimated expected rewards.

The trade-off of utilizing what is known via maximum likelihood and collecting samples from suboptimal actions to get more reliable estimates is called the exploitation-exploration trade-off. Many decision rules trying to balance the two exists and we will look into three of them in the following sections.

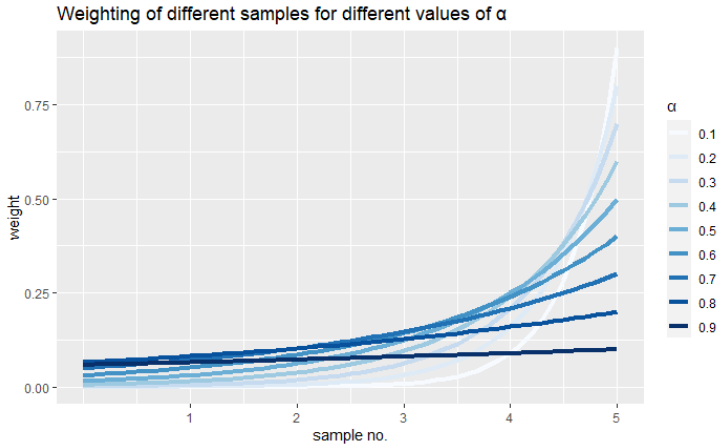

Figure 5: The effect of different values for α for the computation of the weighted mean. On the x-axis we have the order in which the samples were taken, 5 being the most recent. On the y-axis is the weight given to the sample in the computation of the mean. The smaller the α the more weight is given to recent samples (and less to old).

4.1.1 ε-greedy

The ε-greedy decision rule is probably the simplest decision rule one can find that balances exploration and exploitation to some extent. It suggests to select randomly and non-greedily from all the actions with probability ε and greedily with probability $1 - \varepsilon$, i.e.

$\mathbb{P}(A_t = \text{argmax}_a\{q_t^*(a)\}) = 1 - \varepsilon$ and
$\mathbb{P}(A_t \neq \text{argmax}_a\{q_t^*(a)\}) = \varepsilon$. ε is typically small, how small depends on the situation but mostly within the interval $[0.01, 0.1]$.

A disadvantage with ε-greedy is that when it selects actions non-greedily (explores), it selects uniformly at random even though we might have an idea of which kind of actions would be the most and the least profitable to get a reward sample from. For example we may want to sample more from reward distributions of actions from which we have few reward samples, as those estimates still have much uncertainty. We should also be interested in almost greedy actions, i.e., actions having an estimated expected reward which is almost as high as the maximum likelihood pick.

Another drawback is that the probability of sampling a reward from a non-greedy action is the same throughout the training, even though taking samples from different reward distributions might be more important early in the training and

of less importance later, as the estimates then are better.

4.1.2 Upper Confidence Bound (UCB)

Upper Confidence Bound (UCB), is a decision rule taking more samples from reward distributions of actions that may be underestimated, such as actions which we have fewer reward samples from but which still have a quite high estimated expected reward. We may say that the UCB decision rule takes potential into account as it when exploring explores the actions which are the most likely to be underestimated due to finite sampling.

UCB is based of Hoeffding's inequality (Hoeffding, 1963):

Theorem 1 (Hoeffding's inequality)**.** *Let $R_1, R_2, ..., R_t$ be i.i.d. random variables such that $0 \leq R_i \leq 1, i = 1, ..., t$. Then for any $u > 0$*

$$\mathbb{P}(\mathbb{E}[R] \geq \bar{R} + u) \leq \exp(-2tu^2) \tag{3}$$

The theorem gives an upper bound for the probability that the expected value is bigger than the estimated expected value (the mean) plus some constant. That probability is exponentially decreasing with u and the number of variables used to compute the mean, t.

We can think of it as a bound to the probability that the estimated expected reward of an action is underestimated due to finite sampling. Actions with higher probability of underestimation are those we want to take more samples from. We may think of u as a threshold value, indicating how much we underestimate.

We replace $\mathbb{E}[R]$ in Theorem 1 with our notation for the expected reward given action a, $q(a)$, and $\bar{R}$ with $q_t^*(a)$. Furthermore we replace u with $U_t(a)$ as we want it to depend on the action and time. Lastly, we replace t with $N_t(a)$, the number of samples taken from the reward distribution of action a at time t, which gives

$$\mathbb{P}(q(a) \geq q_t^*(a) + U_t(a)) \leq \exp(-2N_t(a)U_t(a)^2).$$

We set $\exp(-2N_t(a)U_t(a)^2) = \epsilon$, we want ϵ to be small as we do not want to underestimate actions. Solving for $U_t(a)$ we get

$$U_t(a) = \sqrt{-\frac{\ln(\epsilon)}{2N_t(a)}}.$$

A common choice of ϵ is $\epsilon = t^{-4}$. This because it is a small value which also cancels the 2 in the denominator. Solving for $U_t(a)$ we get

$$U_t(a) = \sqrt{\frac{2\ln(t)}{N_t(a)}}.$$

We choose our next action according to

$$A_t = \underset{a}{\operatorname{argmax}} \left[q^*(a) + c\sqrt{\frac{2\ln(t)}{N_t(a)}} \right],$$

which is the UCB decision rule. $c > 0$ is a constant controlling our inclination to explore. Bigger c implies a bigger incentive to explore and sample from actions estimated to be suboptimal. Note that we need to begin with taking every action once to avoid division with zero.

The logarithm favours exploration early on but puts the more trust to experience and favours exploitation in the long run as $N_t(a)$ grows much faster.

Recall that for Hoeffding's inequality the random variables needs to be in the interval $[0, 1]$. This will not always be true for our application where the random variable is the reward. This because we will define the reward to be change in performance of our model, which can be negative. However, as long as R_i is in the interval $[a_i, b_i]$ and $b_i - a_i = 1$ the result will be the same. To see that we use the generalized Hoeffding's inequality:

Theorem 2 (Generalized Hoeffding's inequality)**.** *Let $R_1, R_2, ..., R_t$ be i.i.d. random variables such that $a_i \leq R_i \leq b_i, i = 1, ..., t$. Then for any $u > 0$*

$$\mathbb{P}(\mathbb{E}[R] \geq \bar{R} + u) \leq exp\bigg(- \frac{2t^2u^2}{\sum_{i=1}^{t}(b_i - a_i)^2} \bigg) \tag{4}$$

If $b_i - a_i = 1$ the sum will reduce to t and Equation (4) will equal Equation (3). This will be the case for us, as we will be measuring the performance of our model using AUC (see section 6) and AUC is bounded between 0 and 1. To see this, assume that the AUC score of the previous round is y_i. The extremes then occur if we in the next round obtain perfect performance, $y_{i+1} = 1$, or the worst possible performance, $y_{t+1} = 0$. The upper bound of the change in performance is then $b_i = 1 - y_i$, the lower bound is $a_i = 0 - y_i$ and $b_i - a_i = 1$.

A drawback with UCB is that it is not so powerful in non-stationary situations where the need to explore does not diminish in the same way. Although, one

can combine UCB with exponential recency weighting to make it handle non-stationarity a bit better.

4.1.3 Thompson Sampling

Thompson sampling is a decision rule based on Bayes Theorem:

Theorem 3 (Bayes' Theorem)**.** *Let X and Y be two random variables and $\mathbb{P}(X = x) > 0$. Then*

$$\mathbb{P}(Y = y|X = x) = \frac{\mathbb{P}(X = x|Y = y)\mathbb{P}(Y = y)}{\mathbb{P}(X = x)}$$

We may, as in Bayesian inference, consider ϕ to be a parameter and for it to have a distribution of its own. Then

$$p(\phi|x) = \frac{p(x|\phi)p(\phi)}{p(x)} \tag{5}$$

where $p(\phi)$ is called the prior distribution of ϕ, $p(\phi|x)$ the posterior distribution and $p(x|\phi)$ is the likelihood, which may also be written as $l(\phi|x)$. $p(x)$ is found by using the law of total probability, i.e., for discrete variables $p(x) = \sum_\phi p(x|\phi)p(\phi)$ and for continuous variables the sum is replaced with an integral.

Example 4.1. *Let $R|\phi \sim Be(\phi)$ and $\phi \sim Beta(\alpha, \beta)$. If $r_1, ..., r_n$ are samples of R we get*

$$\phi|R \sim Beta(\alpha + r_i, \beta + 1 - r_i).$$

That is, if we have a Beta prior distribution and a Bernoulli likelihood the posterior distribution will again be a Beta distribution. This can easily be derived with Equation (5) and is done in the appendix.

Example 4.1 displays a commonly used pair of distributions. This is because the distributions fit many multi-armed bandit applications and the Beta distribution is a conjugate prior to the Bernoulli likelihood, meaning that the prior- and posterior distribution are belonging to the same distributional family. This is a great advantage, as computing the posterior can otherwise be cumbersome, time consuming and require approximations.

The idea of Thompson sampling is to assume a prior distribution for the probability of a (high) reward and to update the prior distribution as we learn more,

i.e., as we receive more reward samples. The distributional assumptions are used to estimate the expected reward. To include some exploration into the decision rule, we sample from the prior distribution of each action each round and the action with the highest sample is the one we execute. Thompson sampling is conducted according to the Algorithm 2.

Algorithm 2 Thompson Sampling

Let K be the number of different actions and assume the reward R to have a distribution dependent on ϕ, which is distributed according to p. **Iterate (for a predetermined number of iterations or until a convergence criteria is met):**

1. sample $\phi_1, ..., \phi_K$ from $p_1, ..., p_K$
2. select $\phi_i = \text{argmax}_{\phi_j}\{\mathbb{E}[R|\phi_j], j = 1, ..., K\}$
3. sample r_i from $R|\phi_i$
4. update $p_i := p_i(\phi_i|r_i)$

Example 4.2. *Continuation of Example 4.1. Consider having the likelihood function and prior distribution of Example 4.1. One iteration of Thompson sampling is then conducted according to Algorithm 3.*

Algorithm 3 Thompson Sampling with Beta prior and Bernoulli likelihood

Let $R|\phi \sim Be(\phi)$ and $\phi \sim Beta(\alpha, \beta)$, i.e., assume the reward to be binary and therefore Bernoulli distributed and the probability parameter to be Beta distributed. **Iterate (for a predetermined number of iterations or until a convergence criteria is met):**

1. sample $\phi_1, ..., \phi_K$ from $Beta_1(\alpha_1, \beta_1), ..., Beta_K(\alpha_K, \beta_K)$
2. select $\phi_i = max(\phi_1, ..., \phi_K)$
3. sample r_i from $Be(\phi_i)$
4. update $\alpha_i := \alpha_i + r_i$, $\beta_i := \beta_i + 1 - r_i$

The biggest advantage and disadvantage with Thompson sampling are both the distributional assumptions. They provide a framework and give the estimated expected rewards better uncertainty estimates as we get an expression for the variance, which can be utilized in our multi-armed bandit orchestrated federated learning model. However bad distributional assumptions can affect the results in a negative manner. Finding appropriate distributions can be hard in some situations, especially if we want to avoid cumbersome computations for estimating the posterior distributions.

Discounted Thompson sampling

If the application involves non-stationary rewards we may want to make adjustments to the decision rule, discounted Thompson sampling is one such adjustment which was introduced by Raj and Kalyani (2017). Generally we do not know how the reward distributions change, just that they do, and the standard approach in such situations is to increase the impact of the most recent reward samples and to encourage exploration throughout training. For the non-parametric decision rules introduced earlier we can use the exponential recency weighting of the mean, but for Thompson sampling that is not possible as we estimate distributions and not means. Discounted Thompson sampling instead introduces a *disocunt factor* γ discounting the effect of past observations. It also systematically increases the variance of the distributions of the non-executed actions a given round, which enlarges their chances of being executed in the following rounds. The algorithm for the Beta prior - Bernoulli likelihood pair is in presented in Algorithm 4.

Algorithm 4 Discounted Thompson Sampling with Beta prior and Bernoulli likelihood

---

Assume $R|\phi \sim Be(\phi)$ and $\phi \sim Beta(\alpha_0 + S, \beta + F)$, i.e., assume the reward to be binary and therefore Bernoulli distributed and the reward probability parameter to be Beta distributed. Set $S_k = 0$ and $F_k = 0$ for $k = 1, ..., K$. Fix the discount parameter $\gamma \in (0, 1]$ and the initial parameters of the Beta distribution, α_0 and β_0.

Iterate(for a predetermined number of iterations or until a convergence criteria is met):

1. sample $\phi_1, ..., \phi_K$ from $Beta_1(\alpha_1 + S_k, \beta_1 + F_k), ..., Beta_K(\alpha_K, \beta_K)$
2. select $\phi_i = max(\phi_1, ..., \phi_K)$
3. sample r_i from $Be(\phi_i)$
4. update:
   $S_i := \gamma S_i + r_t$ and $F_i = \gamma F_i + 1 - r_t$ and
   $S_k := \gamma S_k$ and $F_k := \gamma F_k$ for $k \neq i$

For $\gamma = 1$ we get the ordinary Thompson sampling decision rule. The discount happens as some of the impact of the old estimates of S and F are reduced as we multiply with a value between 0 and 1. Multiplying both S and F with the same factor, as done in the updates of the algorithm, leaves the mean almost unchanged at the same time as the variance is increased.

5 Neural Networks

This section is based on Goodfellow *et al.* (2016).

This section will go through the very basics regarding the structure and training of a neural network. It will not be necessary to have a good understanding of neural networks to be able to follow the reasoning and to understand the results in this thesis. Yet, some knowledge of neural networks might help the understanding of the very details and the sources of randomness that we encounter in the multi armed bandit orchestrated federated learning model, which will be introduced in Section 8.

A neural network consists of neurons and connections between them. The training corresponds to adjusting the strengths of these connections. The neurons are structured into layers. In its simplest form a neural network consists of three types of layers: an **input layer**, **hidden layers** and an **output layer**.

Each neuron in the input layer corresponds to a dimension of the data, for the application in this thesis we can think of each dimension as an explanatory variable. The output layer gives the output of the model such as a classification or a prediction. In between the input and output layers we find hidden layers. These have no clear interpretation to us, which is why neural network models are often called "black boxes". This makes neural networks suitable for federated learning, as the parameters does not reveal anything about the training data. The purpose of the hidden layers is to find complex structures in data. They are what distinguishes neural networks from other supervised learning models. A simple neural network is illustrated in Figure 6.

Between every two neurons belonging to two consecutive layers there is a linear connection consisting of a weight and a bias, equivalent to an intercept and a slope. To train a neural network is to adjust the strength of the connections, i.e., to adjust the weights and biases.

Every neuron except those in the input layer has an *activation function*. The activation function takes the output of the neurons of the previous layer and their weights and biases as input and "activates" the neuron, shuts it down or something in between, depending on the function. Mathematically this corresponds to giving the neuron a high or low value.

The activation function used in the hidden layers of the neural network used in this thesis is Rectified Linear Unit (ReLU), see Definition 1. It is chosen mainly for practical reasons: it gives constant gradients which speeds up computation and has empirically showed to give good results.

Definition 1 (Rectified Linear Unit (ReLU))**.** *Let* $\boldsymbol{x}$ *be the outputs from the*

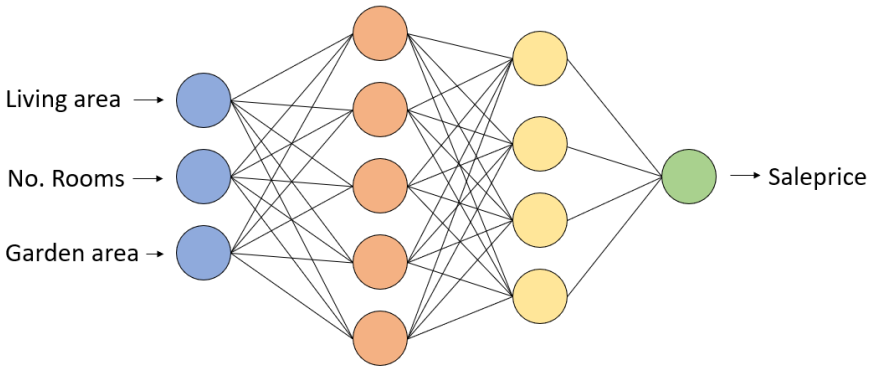

Figure 6: A simple neural network for a predicting the saleprice of a house. The coloured circles are neurons. Each set of same-coloured neurons makes a layer. The blue neurons forms the input layer, each neuron in that layer corresponds an explanatory variable for a saleprice. The orange and yellow neurons belong to two hidden layers. These have no clear interpretation to us but their purpose is to find complex relations between the explanatory variables and the response variable. The green neuron is the output layer which outputs the saleprice.

neurons in the previous layer and $\boldsymbol{W}$ *and* $\boldsymbol{b}$ *be their corresponding weights and biases. Let* $\boldsymbol{s} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$*. We then write the ReLU activation function as*

$$ReLU(\boldsymbol{s}) = \max\{0, \boldsymbol{s}\}$$

The output layer in our networks consist of a single neuron. For regression tasks one can again use ReLU as activation function, but the data set we experiment on has a binary response variable and we therefore have a classification task. This motivates the use of the softmax activation function:

Definition 2 (Softmax)**.** *Let* $\boldsymbol{x}$ *be the outputs from the neurons in the previous layer and* $\boldsymbol{W}$ *and* $\boldsymbol{b}$ *be their corresponding weights and biases. Let* $\boldsymbol{s} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$*. We then write the Softmax activation function as*

$$softmax(\boldsymbol{s_i}) = \frac{e^{s_i}}{\sum_k e^{s_k}}$$

The softmax activation function is a multivariate generalization of the sigmoid function used for example in logistic regression. For binary classification the softmax function reduces to the sigmoid function.

Using the softmax as activation function in the last layer gives output in form of probabilities for the different classes. Our classification is the class with the highest probability.

To get the output we compute all the activation functions from the first layer (the input layer) to the last layer (the output layer), this is called the forward pass. The backward pass consists of computing a loss function measuring the discrepancy between the predicted and the true value and backtracking the error through the network. We do so from the last to the first layer by computing gradients with the chain rule. To do so efficiently we implement SGD. We then update the weights and biases by using the derivatives in the optimization scheme.

For the networks in this thesis we use cross-entropy loss. This is natural since the softmax function outputs probabilities. Recall that cross-entropy is defined as the following:

Definition 3 (Cross-Entropy for supervised learning)**.** *Let $p(y|x,\theta)$ and $q(y|x)$ be two probability distributions with the same support. Y is the response variable, X is the data and θ is the parameters of the prediction model. The cross-entropy of the distribution $p(y|x,\theta)$, the predictions, relative to the distribution $q(y|x)$, the true values is defined as*

$$H(q(y|x), p(y|x,\theta)) = -\mathbb{E}_{q(y|x)}[\log(p(y|x,\theta))].$$

If $p(y|x,\theta)$ and $q(y|x)$ are discrete random variables the above expands to

$$H(q(y|x), p(y|x,\theta)) = -\sum_{y} q(y|x)\log(p(y|x,\theta)).$$

As $q(y|x)$ gives the true values of the response variable Y given the data X we get

$$q(y|x) = \begin{cases} 1, & \text{if } y \text{ is the correct label for the data x} \\ 0, & \text{otherwise} \end{cases}$$

Hence,

$$H(q(y|x), p(y|x,\theta)) = -\log(p(y|x,\theta))\mathbb{1}_{\{y\text{is the correct label for } x\}},$$

where $\mathbb{1}_{\{z\}}$ is the indicator function which equals 1 if z is true and 0 otherwise.

Now consider p in the definition to be the softmax function from Definition 2. This gives us the cross entropy loss

Definition 4 (Cross-Entropy loss)**.** *Let $\boldsymbol{x}$ be the outputs from the neurons in the previous layer and $\boldsymbol{W}$ and $\boldsymbol{b}$ be their corresponding weights and biases. Let $\boldsymbol{s} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ be the output of the last layer. If we use the softmax activation function and s_y is the true label we get the loss*

$$l = -log\left(\frac{e^{s_y}}{\sum_k e^{s_k}}\right).$$

We update the model by utilizing gradient descent, but as neural networks often are big and require very big amounts of training data the computation of the gradients can take a tremendous amount of time and be a bottleneck for the whole system. Therefore we restrain from computing the gradient on all the data at the same time. By definition stochastic gradient descent is to compute the gradient one observation at a time, but the practice is often to split the training data into distinct subsets which we call mini-batches. This is called mini-batch SGD but is often what is implied when saying SGD. By computing the gradients on subsets of the data we are not guaranteed to reach the true gradient, but often we are close enough. Of course, we need to make sure that the data is not in any specific order such that the batches can be assumed to be identically distributed. The steps are shown in Algorithm 5.

Algorithm 5 Mini-batch Stochastic Gradient Descent (SGD)

Denote the model parameters by $\boldsymbol{\theta}$ and the loss function by l.
Fix a learning rate η, a batch size B, the number of epochs E.
for *epoch in [1,...,E]* **do**
  split the data into batches of size B
  for *b in batches* **do**
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla l(\boldsymbol{\theta}; b)$
  end
end

For the neural networks in the experiment we also use batch normalization layers to avoid exploding gradients (that the range of gradient values increases exponentially with every layer) which easily happens in networks with many layers and suboptimal initial parameter values.

The initial values for the parameters of the neural network are set by sampling values from a normal distribution. Hence, we can expect each experiment to have different initial conditions.

The very details and the finesses of the neural networks used for the experiments in this theses are however not that relevant as we are not aiming to find a

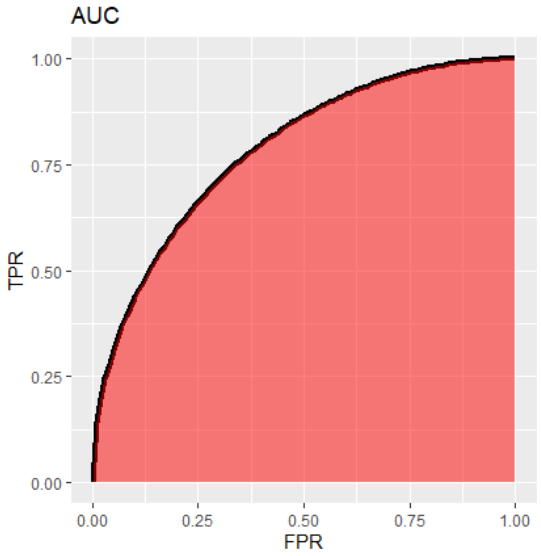

Figure 7: Example of a ROC curve (black solid curve) and corresponding area under the curve, AUC (red area). On the x-axis we have the False Positive Rate and on the y-axis we have the True Positive Rate. For this example AUC = $\pi/4 \approx 0.79$

peak performing model but rather to see how combining models affects the performance.

6 Area Under the Curve (AUC)

This section is based on Juan and Ling (2005) unless stated otherwise.

AUC is a method to measure performance, it is especially useful when having unbalanced data as it measures the model's ability to distinguish between two classes rather than the proportion of correct predictions. It is short for Area Under the Curve, the curve being a ROC curve, which is short for Receiver Operating characteristic Curve. We get the ROC curve by varying the regularization parameter of the model and plotting the True positive rate (TPR) against the False positive rate (FPR), see Figure 7.

We have

$$TPR = \frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false negatives}}$$

$$FPR = \frac{\#\text{false positives}}{\#\text{false positives} + \#\text{true negatives}}$$

TPR is the same as sensitivity and FPR equals one minus specificity. We can also express TPR and FPR in terms of Type I and Type II errors.

We want that if the false positive rate increases (which is bad) the true positive rate should increase more (which is good), meaning that we want the classes well separable, i.e. easy to tell apart. We measure this by computing AUC, the area under the ROC curve. As the axes of the ROC curve, TPR and FPR, have values ranging from 0 to 1 we get the area ranging from 0 to 1 as well. A model which is 100 % correct has AUC equal to 1 and a model classifying everything incorrectly has AUC equal to 0. A model unable to tell two classes apart, where every classification is completely random (50/50), is expected to have AUC equal to 0.5 .

This way of measure performance measures the model's ability to tell cases apart as opposed to accuracy which mainly measures the proportion correctly classified. AUC is preferable to accuracy when we have unbalanced classes, as otherwise we might get a high accuracy from classifying every data point to the biggest class, which is not desirable as we then do not learn anything about what differs between two classes. AUC is mainly suitable for binary classifiers, the generalisation for multiple classes is more complicated and utilizes the pairwise AUC scores (Hand *et al.*, 2001).

7 The Internal Data Set

The internal data set is a data set held by Ericsson relating to telecommunication. The explanatory variables are mainly continuous except for 2 ordinal categorical variables. Negative values exist but no values are missing. The response variable is binary which means that we are doing classification.

The data splits naturally (because of how the data was collected) into 4 workers. We denote a combination by a binary coding, for example the combination 0110 is the combination including worker 2 and worker 3.

The data set has about 40 000 data points (rows) with 153 explanatory variables (columns). When we have access to big amounts of data in each worker we can

Table 1: Proportion of data with label 1

Worker	Training Data	Validation Data	Test Data
1	42%	47%	47%
2	47%	48%	48%
3	31%	29%	31%
4	35%	30%	29%

train a very well performing model on only the local target worker model and there might be no need for federated learning. As we are mainly interested in situations where the available data in the target worker is not sufficient for a good performing model and we need to combine models we use as little as 1% of the data in the training set. For bigger proportions the experiments are very uninteresting as combinations are then avoided and the target worker model alone is favoured. We put 40 % of the data in the validation set and the remaining 59% in the test set. This is a massive amount of data compared to the data in the training set, normally more than half of the data are assigned into the training set. We could have reduced the data set and kept more standard proportions, but we chose to give ourselves the luxury of not having to worry about whether the validation and test data are big enough.

It would have been optimal to use a dataset with more complex structures, but this data set has been used as it is well known within Ericsson and finding a good dataset suiting this method in its current state is time demanding. This is further discussed in the discussion section (Section 10).

We select the data points to the training- validation- and test sets uniformly at random. The data set is unbalanced although not severely so and the balance vary some between workers, see Table 1. Each worker has the same amount of data.

The target worker will always be the same in this thesis (worker 2). This is mainly because the experiments are very time demanding and we have prioritized to conduct extensive experiments for one target worker rather than conducting some experiments for each. The target worker corresponds to combination 0100.

The performance will be measured with AUC.

8 Multi-Armed Bandit Orchestrated Federated Learning (MAB-FL)

The model in this thesis constitutes a novel method to conduct personalized federated learning. We get to the model by generalizing the federated averaging algorithm introduced in Section 3.1.

Recall that federated averaging can briefly be described by the following steps:

Algorithm 6 Outline of Federated Averaging

Iterate (for a predetermined number of rounds or until a stopping criteria is met):

1. Send the parameters of the global model to every worker.
2. Train the models locally on every worker with SGD.
3. Aggregate the models by taking the weighted mean.
4. Make the aggregated model the new global model.

Federated Averaging is taking every worker into account (or some proportion of them if the number of workers is big), but might not be suitable if we wish to make predictions for a specific target worker as the global model becomes very general. This is because in most federated learning applications the workers are often not i.i.d. and can differ much from each other. Instead of Federated Averaging, we want to train a model optimizing the performance on a specific target worker. Note that we wish to train a model optimizing for one given target worker only, not every worker simultaneously as other personalization models do.

Avoiding federated learning entirely and only training a local model on the target data set is one way to do it, but it may also be the case that the available training data on the target worker is not sufficient for the complexity of the model. In that case influence from other workers might be beneficial, just not all of them as in Federated Averaging, or even most of them. We would like to find the appropriate combination of workers which, when we aggregate their model parameters, contribute positively to the performance of the model on the target worker.

Instead of step 3 in Algorithm 6 we would like to do:

3'. Choose the best combination of workers for the performance on the target data set and aggregate those only by taking the weighted mean.

Given the best combination we can also completely disregard the other workers in step 1 and 2 of Algorithm 6. The multi-armed bandit orchestrated federated learning model is our proposed way of implementing this.

8.1 The model

Our model, multi-armed bandit orchestrated federated learning, which will be introduced in this section, proposes a way of finding the optimal combination of workers for a given target worker. The model does this while training global model, which is the personalized model as the machine learns to aggregate the combination which is optimal for the target worker and makes the aggregated model the global model.

Our strategy to find the best combination is to use the multi-armed bandit framework. Selecting a combination of workers to aggregate is an action and the resulting performance increase on the target worker is mapped to a reward. The performance increase is the difference in performance of the aggregated model and the local model on the target worker during the same round. Defining the reward to be the performance increase instead of the performance is a way of standardizing the performance, which is necessary in order to be able to apply the multi-armed bandit framework. This is because we then can imagine that we, to some extent, have a fixed single-state environment where every reward sample happens with the same conditions. This will not be the case if we let the reward be the performance. This, and our way of measuring performance increase, is further motivated in Section 8.2.2.

The rewards will let the multi-armed bandit decision rule learn which combination gives the biggest performance increases, this is the best combination. By executing this combination we should then get the best personalized model. This is the idea of the multi-armed bandit orchestrated federated learning model (MAB-FL), which we can see in Algorithm 7.

Note that we always evaluate the performance on the target worker. When saying "the performance of the global model" or similar we always mean the performance of the global model on the target data set. The global model is the personalized model for the target worker. Optionally we can also after finishing training continue tuning the model locally. We may also train a new model with the best combination held fixed and see whether this model is performing better. It should for stationary reward situations as we then assume that the best combination is the best during all rounds. It is however possible that it is beneficial to occasionally get the influence of a suboptimal combination, to get out of a local minima or for regularization. The regularizing effect comes from that we force the model to be more general by aggregating more models.

Algorithm 7 Multi-Armed Bandit orchestrated Federated Learning (MAB-FL)

Fix the number of epochs per round, E, and the parameters of the decision rule.
Iterate until the global model meets some stopping criteria:

1. Use the decision rule to select a combination of workers to aggregate.
2. Send out the parameters of the global model to the workers in the selected combination.
3. Train the model on the workers in the selected combination with SGD for a E epochs.
4. On the target worker, measure the performance on the validation data.
5. Aggregate the models of the selected workers. Make the aggregated model the new global model.
6. Measure the performance of the aggregated model (the global model) on the validation data on the target worker.
7. Compute the difference of the performances measured in step 6 and step 4 and map this to a reward.
8. Update the decision rule to take into account the received reward.

In Section 8.5 we will perform experiments with the MAB-FL model. The decision rules we will try out are ε-greedy (Section 4.1.1), UCB (Section 4.1.2), Thompson Sampling (Section 4.1.3) and a more extreme decision rule which we will introduce in Section 8.3 which we call Extreme Explore. The Extreme Explore decision rule is different as it is deterministic and does not adapt to experience nor tries to be optimal, but always chooses to try out all combinations and makes the best of them the global model. It should perform the best as it collects the most reward samples but it is also the most expensive in communication costs. We will try two versions of it, Extreme Explore and Bayesian Extreme Explore, where we utilize the estimation techniques of the UCB/ε-greedy decision rule and Thompson sampling respectively. We will try to assume both stationary and non-stationary rewards and use the rewards which will be introduced in Section8.4.

The fact that we train the decision rule during the training of the global model causes two main challenges which we will look closer into. The first main challenge is that the training period of the decision rule is restricted to that of the global model (Section 8.3). The second main challenge is we need to distinguish combination effects from training effects and avoid randomness from the training of the global model to seep into the training of the decision rule as much as possible (Section 8.2). The following sections will focus on the challenges

of the MAB-FL approach and some partial solutions.

8.2 Distinguishing combination effects from each other

In our model we have two sources of randomness relating affecting the estimates of the combination effects. The main source is randomness originating from the different initial conditions (the global model) provided to the local models every round. We also have some randomness coming from the SGD optimization schemes used to train the neural networks. The randomness from different initial conditions every round will by far be the largest contribution to the randomness of the model as these will vary greatly between all rounds. For situations with little data in each worker, the kind of situation for which we believe the method will be the most beneficial, the randomness from SGD may even be negligible. The number of mini-batches used for stochastic gradient descent will then be small and it might even be feasible to use only one batch, giving us a deterministic ordinary gradient descent. We can allow the computations of the gradients to take more time in situations where communication happens seldom, which allows for bigger mini-batches.

8.2.1 Adjusting the number of epochs per round

Distinguishing combinations from each other can be hard if the effects are similar and we have much randomness. The effect we observe of a combination can be tracked back to the local models on the workers. Each round every worker receives the updated global model and trains it to better fit the local data. This means that if we train the local models for a very short period each round, they will all be very similar to the global model of the previous round and hence all be very alike. Consequently, the combination effects will be hard to distinguish.

We need to train the local models until they are sufficiently adapted to the local data so that the models deviate from each other and combinations give different measurable effects. We can even afford some overfitting on the local models as model averaging has a regularizing effect. This is because it forces the two models together, making a more general one. The number of epochs decides the amount of training done on a neural network, more specifically the number of iterations of SGD. This means that the number of epochs per round decide how far we train the local models and how adapted we let them be to the local data.

We perform some experiments for two different number of epochs per round on fixed combinations of workers to see the difference. The experiments are conducted on the internal data set and for each setting we run 10 experiments. Some more about the architecture of the underlying model can be read in the

Table 2: Standard deviation of residuals

Combination	4 epochs per round	6 epochs per round
0100	0.013	0.013
0101	0.020	0.013
0110	0.018	0.020
1100	0.023	0.019
FedAvg(1111)	0.020	0.014

main experiment section, Section 8.5.

When comparing the results, we are interested in the best performance of the global model, how well the best combination can be distinguished and how many rounds it takes as the communication costs grow with the number of rounds.

The learning curves for the global model when using 4 and 6 epochs per round and different combinations are shown in Figure 8. From this figure, is seems that the best performances of the global model are very similar for the tried numbers of epochs per round, although the very best performances are with 6 epochs per round. The main difference is that with more epochs per round we finish training with fewer rounds which lowers our communication costs. The curves also seem to be a bit more separated for 6 epochs, which should make the effects more distinguishable.

Figure 9 illustrates how well the combinations can be distinguished from each other for 4 and 6 epochs per round. From this figure, it seems that for 6 epochs per round the best combination (0110) is more distinguishable, the gap to the other combinations is wider. Disadvantageous combinations (0101 and FedAvg which is 1111) also perform better in the first half when training with 6 epochs.

Even though more epochs should in general result in higher variance, as the models become more adapted to the local data, we can see in Table 2 that the standard deviation is the same or less for all combinations except one when moving from 4 to 6 epochs per round. Each of the values in the table comes from 10 experiments with the same settings and 10 and 15 rounds respectively, as that seems to be the number of rounds required until peak performance.

That the variance does not grow with more epochs may be because 6 epochs might have been sufficient for the local models to converge to a local minimum. 4 epochs might only be enough to make the local models differ some from each other, but not enough to reach a local minimum and local convergence and therefore the results are more widely spread.

We check the normal distribution assumption of the residuals to the mean in Figure 10 and Figure 11. We see that they are far more well met for 6 epochs

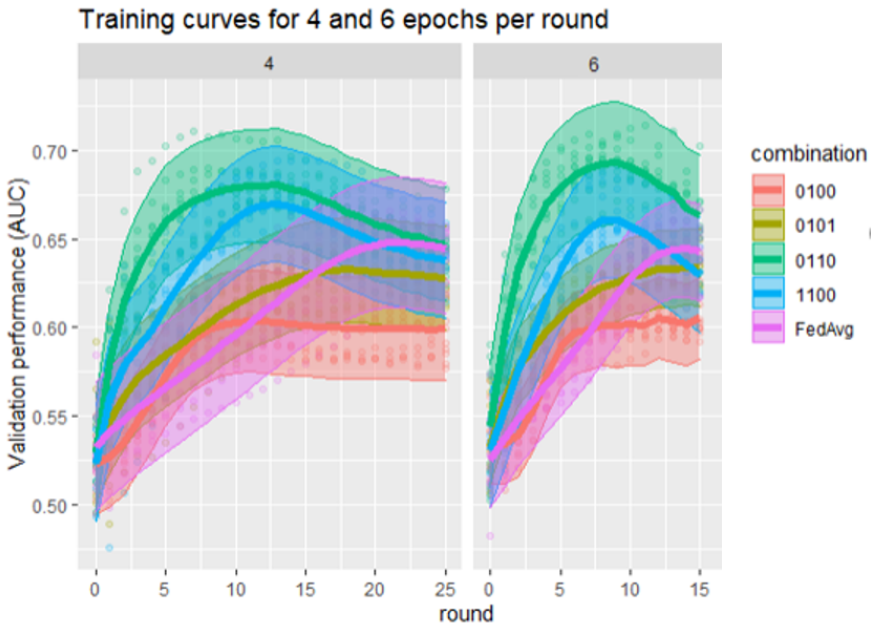

Figure 8: Comparison of the learning curves when training with 4 or 6 epochs per round and holding the combination fixed throughout training. x-axis shows number of rounds of training and the y-axis the validation performance on the target worker. Each color is the mean performance of a combination with a 95% confidence interval, estimated assuming normal distributed deviations from the mean. 10 experiments are performed for each combination and number of epochs per round. The best performance for the best combination (0110) seems to be a little higher for the training with 6 epochs, although not significantly. Training with 6 epochs makes the global model reach better performance in less rounds and the means seem to be further apart and easier to distinguish. It seems that combination 0101 and 0100 has a slightly increasing trend at the 15th round for 6 epochs per round, but the increases during the later epochs are less than 0.01.

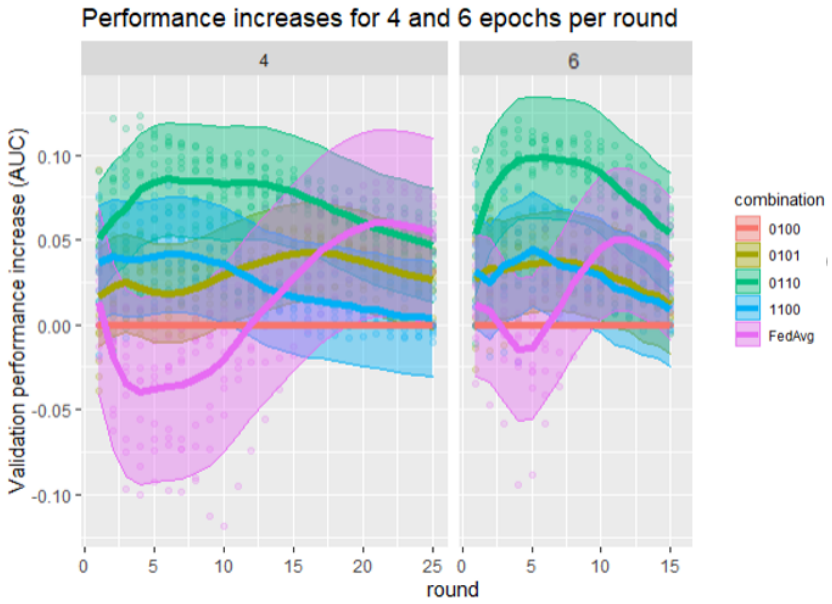

Figure 9: Comparing the performance increases per round of the global model when training the global model on 4 and 6 epochs per round. Performance increase is measured as difference in performance of the aggregated model with the model on the target worker. Combination 0100 contains only the target worker. The x-axis is the number of rounds and the y-axis measures validation performance increase, measured with AUC. The coloured areas are 95% confidence intervals estimated assuming normal distributed deviations from the means. We see that for 6 epochs per round the best combination (0110) distinguishes itself a little more than for 4 epochs per round, the gap to the second to best combination is wider. Federated Averaging is the only model is which having a mean performance increase which is worse than the target model at some time period. That is, around round 5 the pink line is below the red line. We see that Federated Averaging seems to have a bad start but outperforms the target worker model during later rounds. The best model (0110) outperforms the target worker model from start and in the early rounds. We see negative slopes for all combinations except the target worker which we compare with at the later rounds, this is because the round with the peak performance has been passed (see Figure 8) and the models start to overfit.

Figure 10: Scatter plot for the residuals to the mean when training with 4 epochs per round for different combinations held fixed throughout training. The x-axis gives the round of training and the y-axis gives the residual to the mean. Each plot is one combination. The red solid line is the mean and the blue area is the 95% confidence interval, fitted by assuming normally distributed residuals. The residuals have some paths and do not seem like noise.

per round. For 4 epochs per round there are clear differences between different experiments. For 6 epochs per round the residuals seem far more like normal distributed noise. In the Appendix are complementary QQ-plots. In Figure 10 there are some paths. These might have formed as the local models are not trained very far and do not deviate much from each other. Therefore there is no "conflict" when the models are aggregated and the transitions between rounds are smooth.

Experiments with fewer than 4 epochs per round require more rounds for a good performance on the global model and the combination effects to be less distinguishable. More than 6 epochs should either cause overfitting or make the training require the same number of or even fewer rounds. That would however leave us with very few rounds of training, very little data and maybe too easily distinguishable combinations which would make the rest of the thesis uninteresting, so we continue with 6 epochs per round for the rest of the experiments.

A good alternative to a fixed number of epochs per round should be to train the local models until they overfit slightly and to implement early stopping to prevent severe overfitting.

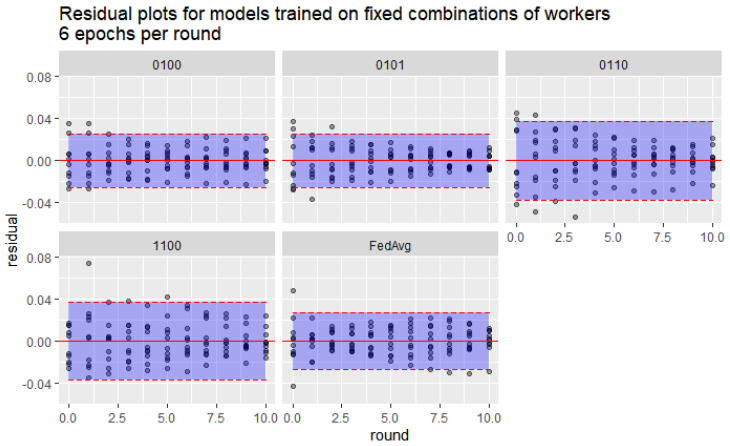

Figure 11: Corresponding figure to Figure 10 but for 6 epochs per round. The residuals seem more randomly distributed and even.

8.2.2 Isolating the combination effect from the training effect

We can vary the amount of variance associated with the estimates of the reward by adjusting how we measure an increase of performance. We need to base the rewards on performance increases and not performances as we wish to utilize multi-armed bandit ideas, which requires a single state environment. This means that every time we execute an action we do so with the same settings and conditions. If we let the reward be the performance we do not satisfy the single state environment assumption as the performance then has an increasing trend due to training. This gives the rewards a dependence on the phase of training, which we do not want. Performance increases however fits the assumption better, but how we measure increases does also play an important role. We compare two different ways of measuring performance increase in Figure 12.

Measuring the performance increase by comparing with the performance of the previous round means that training has occurred between the two measurements. Consequently, we will have problems distinguishing the effect of a combination from that of training, this adds variance to our estimates of the combinations' effects. All uncertainty adds to the variance. We also get an unwanted strong dependence of the last round. For example, if the combination used for the current round differs from the one of the previous round we will have problems knowing if we observe an increase because the previous combination was inappropriate, because the current one is advantageous or both. That is, the reward of one round depends on the combination executed the previous round.

We wish to isolate the effect of the combination used for the current round

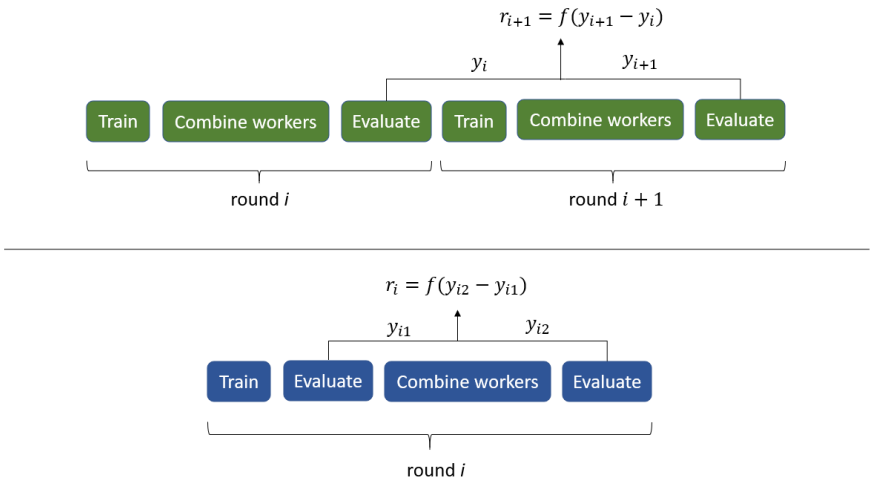

Figure 12: Two different ways of measuring increase of performance. In the upper part of the figure (green) we compare with the performance from the last round. y denotes performance and f the function mapping the performance increase to the reward r. In the lower (blue) part we instead compare the performance with that of the target worker model during the same round. This makes it possible to assume that the rewards are relatively independent and that each reward only depends on what happened during the current round. y_{i1} denotes the performance of the target worker model and y_{i2} the performance of the aggregated model, i.e. the combination.

as much as possible. To get rid of the effect of training we can compare the performance of the aggregated model with the performance of the target worker model just before the aggregation of models was done. By measuring the performance increase compared to the performance on the target worker during the same round we also get rid of most of the dependence of the previous round. When measuring performance increase compared with the target worker it is more reasonable to assume the reward of the current round to be relatively independent of performances of other rounds, i.e., we can assume that we satisfy the Markov property:

$$r_i = f(y_{i2}|y_{i1}, y_{(i-1)2}, y_{(i-1)1}, ..., y_{12}, y_{11}) = f(y_{i2}|y_{i1})$$

Where y_{i2} is the performance of the aggregated model round i and y_{i1} is the performance of the target worker model round i. r_i is the reward round i and f is the function mapping a performance increase to a reward. This is illustrated in the lower panel of Figure 12.

This property gives the environment a single state as there are no dependences of what happened other rounds and we are more in accordance with multi-armed bandit model assumptions. Comparing combination effects with each other should be easier as their estimated effects are not dependent on each other. It is reasonable to assume that this holds for most of the training of the global model, with possible exceptions for the very beginning and end of training where performance increases are very big and very small respectively. That the rewards are independent of the previous round is thus not entirely true as the rewards depend on the phase of training. However, they are relatively independent, as they are not depending on the previous round directly, and enough so to assume a single state environment.

In Figure 13, we see how much more distinguishable the different combinations become when comparing with the model on the target worker. Also we get less outliers which is a big advantage as outliers can have a major impact on the expected rewards. We get outliers as when comparing with the performance of the previous round it is possible that two consecutive combinations are two extremes. That is, the combination of one round was very inappropriate and the combination of the next round is was very advantageous. By always comparing with the same worker we remove that scenario. From now on, a "performance increase" will always refer to the difference in performance of the aggregated model and the target worker model during the same round.

Additionally, if the target worker alone is the best combination, which is often the case, we will get an indication that that may be the case rather quickly by measuring increases compared to the target worker.This will also not result in any extra communication costs, as the performance on the target worker

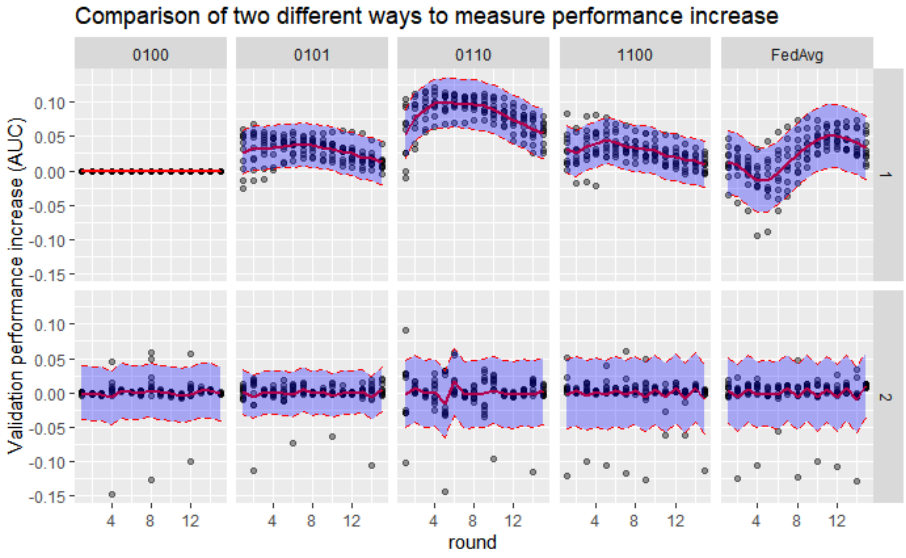

Figure 13: Comparing the effects of the two different ways to measure performance increase (AUC) on the internal data set when the combination is held fixed and we train with 6 epochs per round. We perform 10 experiments per setting, an experiment is the training of a global model. Some more about the architecture of the underlying model can be read in the main experiment section, Section 8.5. The red line is the mean of the performance increases the given round for all 10 experiments and the blue area is the 95% confidence interval, assuming normal distributed deviations from the mean. Each column is a combination. A 1 on position i means that worker i is included in the combination. In FedAvg (Federated Averaging) every worker is included. The top row (1) is measuring performance increase as compared to the performance on the target worker, as showed in the lower scheme of Figure 12. The bottom row (2) measures performance increase as compared to the last round, as showed in the upper scheme of Figure 12. It is clear that in the top row it is easy to distinguish the combinations, we have less outliers and more stable and different means. For example, the mean of one combination is in many cases outside the confidence interval of other combinations, meaning that the effects are significantly different for several rounds. In the bottom row we have many outliers even though we always compare with the same combination, although we have a training round in between the measurement. This is what causes the outliers, which have a huge impact on the estimates of the means.

is computed after the worker has finished training and can be communicated together with the local model update. The target worker model should always be included in every combination and therefore also in every communication round.

8.3 The shortage of samples and how to get more

The training of the multi-armed bandit takes place during the training of the global model. This means that when the training of the global model is completed we cannot train the multi-armed bandit any longer as we will not get more samples from the reward distributions. It is therefore not sure that we will be able to say with certainty which is the best combination when the training is put to an end and consequently we may not know if we have trained the best global model. This is a stark contrast to the ordinary multi-armed bandit or reinforcement learning situation, where we often can continue to take samples (although not 'for free') until we are satisfied and have enough data to conclude which action is optimal.

It is possible to continue training past the round where we observe the best performance in order to get more information about which is the best combination, but the risk of overfitting is high. Especially if one only has a small data set. This is illustrated in Figure 14, where we can see that the performance of the global model decreases rather than stabilizes when continuing past the round with the best performance. It is likely that training past the best performance of the global model results in incorrect estimates of which is the best combination. This is because the combinations that had high estimated rewards when the best performance was reached will be the ones which will be executed the most in the following rounds. However, then the performance decreases because of overfitting and the rewards will consequently be low and have big impact of the estimates of combinations with previously high estimated expected rewards.

This leads to the additional challenge of finding the appropriate stopping time of training. As that is not a problem unique for the MAB-FL model, but one we often face when having models with tendencies to overfit, we will not go further into this challenge here. Although, we will discuss it shortly in the discussion (Section 10).

8.3.1 Reducing the number of combinations to consider

The more combinations we have to consider the fewer reward samples we will be able to take from each combination in some period of time or number of rounds.

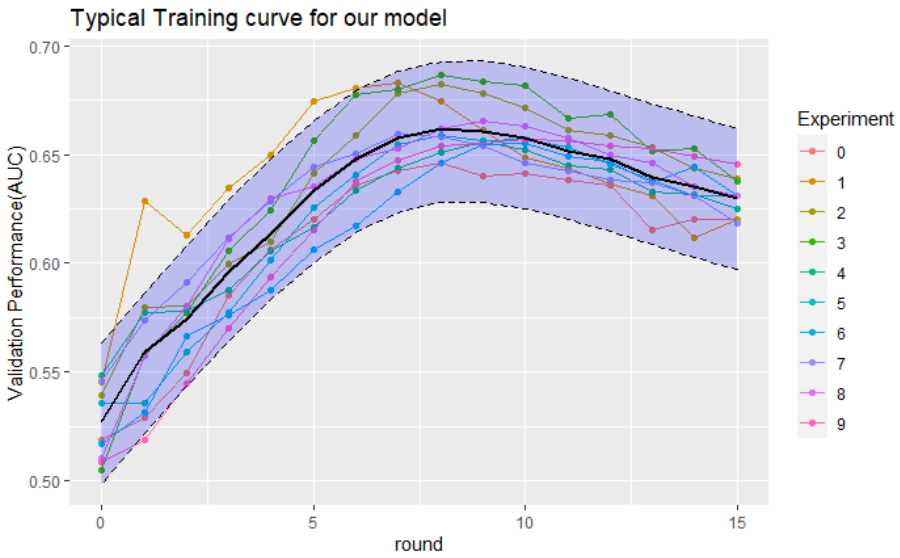

Figure 14: A quite typical learning curve for our model, coming from experiments with combination 1100 held fixed and training is made for 6 epochs per round. On the x-axis we see the number of rounds of training of the model and on the y-axis the performance on the data set in the target worker, measured with AUC (Section 6). Each colored line corresponds to an experiment and the black line is the mean performance. The blue filled area is the 95% confidence interval, estimated assuming normal distributed deviations from the mean. After reaching its peak the performance goes down. This illustrates that we have a limited time period to collect samples from the reward distributions, i.e., the time until the global model peaks in performance.

Think of a combination as a binary series $w_1, ..., w_k$ where $w_i = 1$ if worker i is included in the combination and zero otherwise. Let T be the index of the target worker, then we always have $w_T = 1$. For all other indices, we choose $\mathbb{P}(w_i = 1) = \mathbb{P}(w_i = 0) = 1/2$. Considering every possible combination of workers results in 2^{k-1} possible combinations where k is the number of workers and we always include the target worker. Every worker can either be included or not, therefore the 2. That is a lot of combinations, and the amount is growing exponentially with the number of workers. The number of rounds required to train the global model probably does not increase to the same extent, so with more workers comes less reward samples per combination.

We measure the overlap as the number of workers, excluding the target worker, which are included in two combinations. For example the combination 0110 and the combination 1110 where index 2 is the target worker have overlap equal to 1 as worker 3 is included in both of them (and we do not count the target worker).

Let X measure overlap and let X_i denote the overlap at position i. X_i is an indicator function taking the value 1 if the two combinations both have the value 1 at position i and 0 otherwise. We get that $X = \sum_{i \neq T} X_i$.

The expected overlap for two different randomly chosen combinations is

$$\begin{aligned}
\mathbb{E}[X] &= E\big[\textstyle\sum_{i\neq T} X_i - \mathbb{1}\{\sum_{i\neq T} X_i = k-1\}\big] && (6)\\
&= E\big[\textstyle\sum_{i\neq T} X_i - \prod_{i\neq T} X_i\big] && (7)\\
&= \textstyle\sum_{i\neq T} \mathbb{E}[X_i] - \prod_{i\neq T} \mathbb{E}[X_i] && (8)\\
&= \textstyle\sum_{i\neq T} \mathbb{P}(X_i = 1) - \prod_{i\neq T} \mathbb{P}(X_i = 1) && (9)\\
&= \textstyle\sum_{i\neq T} \frac{1}{4} - \prod_{i\neq T} \frac{1}{4} && (10)\\
&= \frac{k-1}{4} - \left(\frac{1}{4}\right)^{k-1} && (11)\\
&= \frac{k-1}{4} - \frac{1}{4^{k-1}}. && (12)
\end{aligned}$$

In (6) we need to subtract the situation that the combinations are identical as we want to measure the overlap for two *different* randomly chosen combinations. In (7) we use that X_i is an indicator function and that the non-target worker indicators sums to $k-1$ implies that they all equal 1 and that the product should be 1. In (8) use that as X_i and X_j are independent for all values of i and j.

We get the proportion overlapping (non-target) workers to be

$$\frac{\mathbb{E}[X]}{k-1} = \frac{1}{4} - \frac{1}{(k-1)4^{k-1}},$$

which goes towards $1/4$ as k grows big.

As two random combinations can be expected to have almost 25% overlap for $k > 3$ there will be a lot of similar combinations, probably also with similar performances. If we instead wish to measure overlap as the number of indices which have the same value in both combinations, we need to replace the $1/4$ in the computations with $1/2$ and the expected fraction of overlap instead becomes almost 50%. However, measuring overlap only in the included workers makes more sense as we wish to measure the effect of a combination, which is caused by the included workers only.

It is a hard task to distinguish combinations with big overlap, especially with very few samples of the effect of every combination. Therefore we continue with combinations consisting of only pairs of workers, a pair being the target worker and one other worker. We then get the expected fraction overlap to equal zero and we are left with far less combinations to consider. The number of combinations to consider then grows linearly with the number of workers.

If two different pairs both seem to perform good we may make a bigger combination with those and run the experiment again, but we avoid bigger combinations unless given strong reasons to consider them.

8.3.2 Producing more samples with Extreme Explore

Having fewer combinations to consider enables us to get more reward samples from the reward distributions for every considered combination. However it still might not be enough. This is, as already mentioned, not an ordinary multi-armed bandit situation, but that should not only be seen as a restriction but also something that we can take advantage of. One big advantage is that we are not limited to evaluate only one combination (try out one action) per round. Actually, we can evaluate every combination every round of training and choose the best performing model on the validation data to be the new main model. We call this deterministic decision rule Extreme Explore, it is illustrated for a simple situation with three workers where we try out pairs only in Figure 15.

The Extreme Explore decision rule should give the best results as it collects the most information, but it is more expensive in communication costs as we involve more workers every round. However for a small number of workers it need not be as expensive as one might think, especially if one is not counting the amount of data sent between the workers and the server but considers every communication

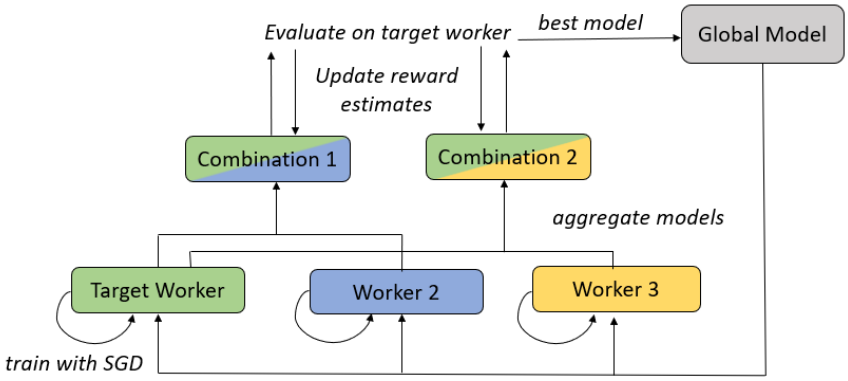

Figure 15: Scheme of the Extreme Explore decision rule for a situation with three workers and only pairs are evaluated. The global model is sent to all workers where it is trained on the local data with SGD. Two combinations of workers including the target workers are possible. The local models in each pair are aggregated by averaging. Each aggregated model is evaluated on the target worker, the model with the highest performance is made the new global model. The estimates of the expected rewards are updated.

to be as expensive. In the computations below we will only count the number of communications and treat each to be as expensive.

Every information exchange between the server holding the global model and a worker counts as communication. For the multi-armed bandit orchestrated federated learning model (MAB-FL) only evaluating pairs sending out the global model to the target worker and the other worker in the selected pair counts as 2 communications. Transmitting the updated models from the workers to the server where they are then aggregated corresponds to 2 communications. Sending the aggregated model to the target worker where it is evaluated counts as one communication and so does sending back the reward. Thus, the simple MAB-FL model with pairwise combinations has 6 communications per round.

If we consider every possible combination including the target worker and have k workers in total the expected number of included workers in a combination is $(k-1)/2+1$ as each non-target worker is included with probability $1/2$ and the target worker is always included. This gives us instead an expected number of $((k-1)/2+1)*2+2 = k+3$ communications per round for the MAB-FL model.

MAB-FL with Extreme Explore always sends the global model to every worker or every relevant worker, no matter the values of the rewards of the previous round. Therefore we do not need to do the two last communications, where the combination is evaluated, directly but can delay the evaluation to next

round. We can then send the models to evaluate to the target worker next round together with the global model, and the delayed rewards are sent back to the server together with the updated model from the target worker. If we delay the rewards, we need to pick the model to be the new global model based on the reward estimates from the last round. However, that should not be much of a problem, especially if we train for several rounds.

For k workers this results in $2k$ communications per round and there is no difference whether we only evaluate pairs or all possible combinations. Consider however that we only evaluate pairs and that we have 6 workers. The ordinary MAB-FL model then still has 6 communications per round, but the MAB-FL model with Extreme Explore has 12 communications per round. The communication cost might be doubled, but we evaluate 5 more combinations, which is five times more. The double cost for five times the information means we pay less than half the price per evaluated model. The difference becomes even less with MAB-FL considering combinations of all sizes. For 6 workers we then expect of 9 communications per round for evaluating 1 combination.

Extreme Explore collects far more information per round and thus can be expected to select better global models each round. Therefore it is likely that it reaches a good performing global model faster than the ordinary MAB-FL. This means that training requires less rounds which makes the Extreme Explore even cheaper.

Furthermore, we might come to the conclusion during training that some combination is significantly worse, we can then exclude it from the model and save 2 communications per round for the rest of training.

An additional advantage with evaluating several or all combinations on each round is that the estimates of the effects of different combinations in relation to each other should be more precise. This is because the effects may change with the number of round finished, but also because we get the rewards by comparing the performance of the aggregated model with the model on the target worker. Therefore we get a dependence on the model on the target worker in a given round. If we evaluate every relevant combination every round, the reward depends on the same target worker model and the combinations are evaluated with exactly the same conditions. This should give the effects of different combinations with less variance and make them easier to distinguish.

When applying Extreme Explore we may choose to either use the non-parametric way of computing the estimated expected reward as done in ε-greedy and UCB or the parametric Bayesian approach as in Thompson Sampling.

8.4 Rewards

After we execute an action by choosing a combination to aggregate, we measure the effect the aggregated model has on the performance of the global model. This is mapped to a reward. The reward is a feedback to the decision rule, which is deciding which action to execute depending on the estimates of the expected reward. The sampled rewards are stochastic, the sources of randomness and how we can reduce their effect on our estimates was described in Section 8.2.

We want the performance of the global model to depend on the aggregated combination of workers but do not want the opposite: that the effect of a combination is dependent on the performance of the global model. That would make it harder to evaluate the effects of different combinations. Therefore we define our rewards based on the improvement of performance compared to the performance of the local model of the target worker in the same round. In that way we isolate the improvement caused by a combination from the effect of training as much as possible. This is further explained in Section 8.2.2. Computing the percentage performance increase, i.e., the performance increase as described above divided with the performance of the local target worker model in the same round, has been considered as well although with worse results. It favours early rounds as the denominator (the AUC performance of the target worker model) is then small.

There is no guarantee that the rewards are stationary. On the contrary, it is likely that we have non-stationary rewards. This is because the rewards depend both on the local target worker model and the aggregated model (the combination), and if they improve with different rates we will see a structural change in the rewards. However, our situation is eased some by the fact that we do not care about estimating the rewards correctly as much as correctly estimating which is the best combination. Therefore some bias in the expected reward estimates poses no problem as long as all estimates are somewhat equally biased.

We will try out some techniques to adjust for non-stationary rewards such as exponential recency weighting of the mean and discounted Thompson sampling. Another way to compensate for non-stationarity is to add Extreme Explore-rounds. During a such round a reward sample is to be taken from the reward distribution of every action. One could make every round an Extreme Explore-round with some probability or make such a round every i:th round, where i may be fixed or depend on the number of rounds completed so far.

We will try out the following rewards:

Let i_t be the performance increase associated with a combination and r_t the reward at time t.

If we relate performance to accuracy (higher values indicate improved performance) we have

Reward type 1 (Increment):

$r_t = i_t$

Reward type 2 (Binary increment):

$$r_t = \begin{cases} 1, \text{if } i_t > \epsilon \\ 0, \text{else} \end{cases}$$

If we relate performance to loss (smaller values indicate improved performance) we need to replace i_t with $-i_t$.

Thompson sampling will only be used with binary rewards as it fits well with having a Beta prior and Bernoulli likelihood. We will try different values for the threshold. UCB and Thompson sampling will be evaluated with both. We will run experiments assuming both stationary and non-stationary rewards to see what difference it makes on the considered data set.

8.5 Experiments

The purpose of these experiments is to see the effects of different decision rules and assuming non-stationary rewards on the considered internal data set.

The underlying neural network model consists of seven hidden layers: four ordinary linear layers with three batch normalization layers positioned in between. They are chosen as they give a fairly complex decently performing model. Recall that we are not looking for an excellent underlying model architecture, but just a model which leaves room for improvements that we can evaluate the MAB-FL method on. The batch normalization layers stabilizes the gradients and make the training more stable. We use the ReLU activation function for all layers except the output layer where we use softmax. The training is done with SGD. Each experiment, the parameters are initialized by sampled values from a normal distribution, which make each experiment unique.

For every setting considered we run 10 experiments because of the random elements. An experiment is to train the personalized global model. One round is an iteration of the MAB-FL model. Before round 0 all the models have been trained for the predetermined number of epochs per round already. We do that because the initial global model probably is very bad and some extra training is necessary for the models on the different workers to adapt to the local data and

Table 3: Experiments with decision rules

	Parameters	
Decision rule	Stationary rewards	Non-stationary rewards
ε-greedy	ε (0.1, 0.2, 0.5)	ε (0.1, 0.2, 0.5) α (0.25, 0.50, 0.75)
Thompson Sampling	ϵ (0.01, 0.025, 0.05)	ϵ (0.01, 0.025, 0.05) γ (0.50, 0.75, 1)
UCB (continuous rewards)	c (0.01, 0.05, 0.1)	c (0.01, 0.05, 0.1) α (0.25, 0.50, 0.75)
UCB (binary rewards)	c (0.05) ϵ (0.025)	c (0.05) ϵ (0.025) α (0.75)
Extreme Explore	-	-

distinguish themselves. The performance is measured with AUC. The decision rules and parameter values for which we run the experiments are in Table 3. The interpretations for the parameters can be found in the decision rules' respective sections.

Our main focus is ε-greedy, UCB with continuous rewards and Thompson sampling. We want to investigate how the performances are affected by our assumptions about whether the rewards are non-stationary or not. We are also interested in how the decision rules perform relatively to each other. For that comparison we also involve UCB with binary rewards and Extreme Explore with both continuous and binary rewards. The parameters for UCB with binary rewards are decided based on the experiments with UCB with continuous rewards and Thompson Sampling.

We begin to compare the best performance the global model reaches in each experiments for ε-greedy, UCB with continuous rewards and Thompson sampling. These experiments are illustrated with boxplots in Figure 16, 17 and 18.

We can see that in our experiments it often does not matter for the best performance whether we assume stationary or non-stationary rewards, except for ε-greedy with $\varepsilon = 0.2$ and somewhat for UCB with $c = 0.05$. Tuning the parameters of the decision rule seems to be more important.

In the Appendix are complementary figures for the experiments depicting the learning curves and the estimated expected rewards for each round, see Figure 23 to Figure 28. These are mostly very similar but some patterns can be seen. $\alpha = 0.75$ may make the best combination more distinguishable. Discounted Thompson sampling makes the best combination much less distinguishable and both 0.025 and 0.05 make good threshold values. This is somewhat surprising as from Figure 9 it seems like 0.025 is a rather low threshold.

Figure 16: The best performance for each experiment for 10 experiments per setting with the ε-greedy decision rule. The y-axis gives the highest test performance reached during an experiment. Performance is measured in AUC on the target worker. Each point represent one experiment and the boxplots illustrate the spread of the values. The black lines inside the boxes are the medians. ε is the parameter regulating the exploration probability. Higher values give higher probabilities. The colours represent different values of α. "stationary" indicates that we assume stationary rewards and have no α parameter, the parameter in the exponential recency weighting. Smaller values give more weight to recent samples. When we assume stationary rewards there is greater variance in the best performances, see the sizes of the purple boxes compared to the ones of other colours. For $\varepsilon = 0.5$ the best performances are very similar when assuming stationary rewards and not, but for $\varepsilon = 0.2$ there are notable differences. Assuming non-stationary rewards yields higher performances for all considered values of α, but the very best performances are reached with $\alpha = 0.75$ (turquoise). For $\varepsilon = 0.1$ the medians are very alike.

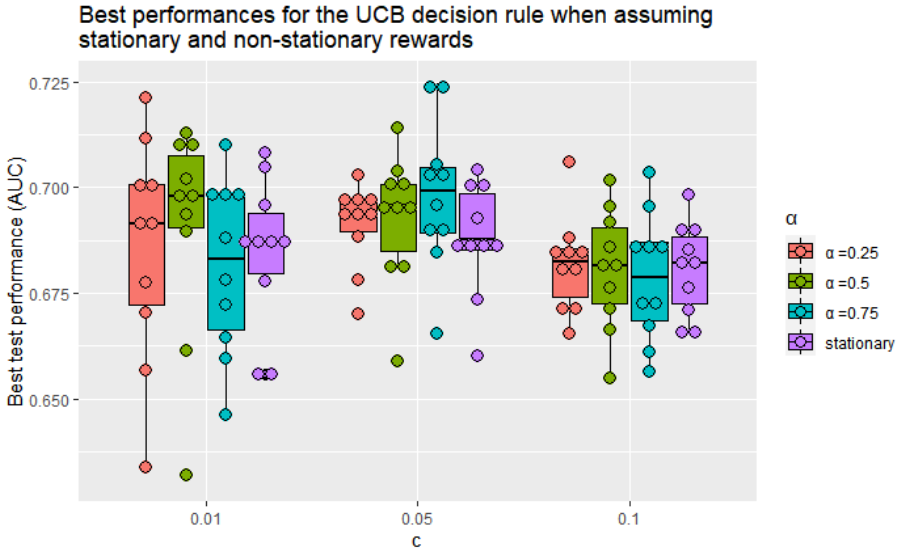

Figure 17: The best performance for each experiment for 10 experiments per setting with the UCB decision rule. For a description of the Figure, see the caption of Figure 16. c is the parameter regulating the exploration incentive. Higher values gives more exploration incentive. The colours are different values for the parameter in the exponential recency weighting, α. Smaller values give more weight to recent samples. "Stationary" means that we use the ordinary weighted mean For $c = 0.05$ and $c = 0.1$ there is very little differences, but for $c = 0.01$ the green and purple boxes are notably smaller. However, they still have negative outliers. From this plot it it seems like c has a stronger effect than α. For different values of c we see that the best performances are better, the medians are among the highest and the negative outliers have higher values. The effect of α seems to be quite small, although the combination $c = 0.05$ and $\alpha = 0.75$ seems to be optimal.

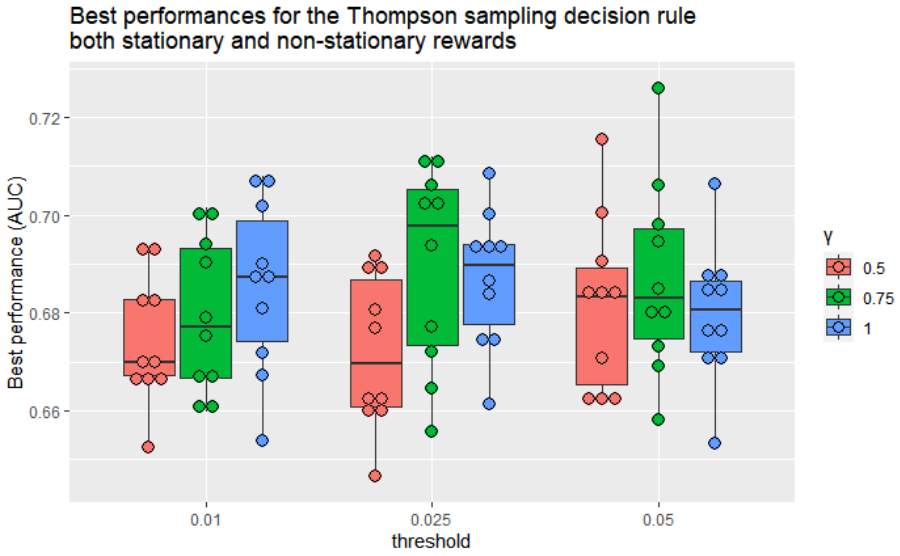

Figure 18: The best performance for each experiment for 10 experiments per setting with the Thompson sampling/Discounted Thompson sampling decision rule. For a description of the Figure, see the caption of Figure 16. Threshold is the reward threshold. Higher values means that bigger performance increases are required for the reward to equal 1. The colours are different values for the discount parameter γ in discounted Thompson Sampling decision rule. Smaller values give more weight to recent samples, $\gamma = 1$ is ordinary Thompson sampling. The threshold values seem to have little importance as the boxplots are rather similar, although the best performances are the best when the threshold equals 0.05. Which is the best value of γ seems to depend on the threshold. The best combination of parameter values seems to be $\gamma = 0.75$ and the threshold set to 0.025, as the median is then the highest. However, $\gamma = 0.75$ and the threshold set to 0.05 has one really high value and it would be interesting to run some more experiments to see if yields more performances above 0.72.

We select the best settings for each decision rule and plot the fitted splines of the learning curves in Figure 19. Extreme Explore is different for continuous and binary rewards as it uses the rewards to decide which model should be the new global model, and for binary rewards there are sometimes "ties" where more than one combination gets a reward equal to 1. Extreme Explore performs in the top as expected, but UCB with binary rewards surprisingly performs better than with continuous rewards. This is surprising as continuous rewards convey more information.

We look closer into UCB with reward type 9. Figure 20 shows the learning curves and the combinations executed each round for all experiments. We can see that 1100 is executed for some rounds in all the experiments, even though 0110 is optimal. Extreme Explore, for example, chooses 0110 for more than 90% of the rounds. That period with 1100 must however be beneficial for the model, as it reaches very high performances. Maybe using different combinations can help with avoiding local minima.

9 Conclusions and Discussion

For the data set used for the experiments with 1% of the data in the training set, we found the personalization to be successful. In this thesis a successful personalization is when the personalized model performs better than both Federated Averaging and a model trained on the target worker only, i.e., a model trained without any federated learning techniques. We can see in for example Figure 8 how the combination 0110 was better than both the target worker model (0100) and federated averaging. When we increased the amount of data in the training set, the personalization was no longer successful as the best model was then the one trained on the target worker only. Those experiments are left out of the thesis as they are of little interest. Based on these results, we conclude that the current state of the method is applicable when we have a quite small amount of data in the target worker. This is reasonable as the need for extra information is the strongest when we have little information available and tend to overfit. When we have more training data, the local models performs quite well by themselves and the regularization from model averaging seems to be too strong. The global model becomes too general and the target worker model is the best model. In the next section (Future work) we discuss how one might refine the model to extend its application to situations with more data.

Many other personalization algorithms focus on tuning the global model locally on each worker to obtain personalized models. This should work well if there are plenty of local training data available, but not as good in the opposite situation, as the local training data is then insufficient to make complex enough local models. This is the situation where the MAB-FL model considered in

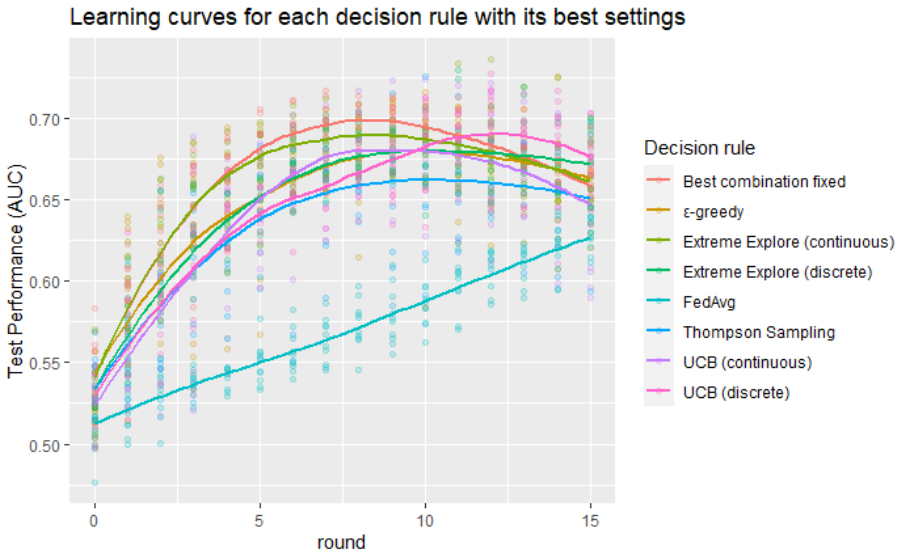

Figure 19: Fitted cubic splines representing the learning curves for all different considered decision rules. The parameters of each decision rule are the ones that seemed optimal in the earlier experiments. The y-axis measures test performance in AUC of the global model. The x-axis is the round of training and each colour is a decision rule. The decision rules, including parameters are the following: ε-greedy ($\alpha = 0.75$, ε=0.2), UCB (continuous rewards, $\alpha = 0.75, c = 0.05$), Thompson Sampling ($\gamma = 0.75$, threshold $= 0.025$), UCB (binary rewards, $\alpha = 0.75, c = 0.05$, threshold $= 0.025$), Extreme Explore (continous rewards), Extreme Explore (binary rewards, threshold $= 0.05$). The best combination (0110) and federated averaging (FedAvg) are not decision rules, but are included for comparison. The best decision rules are Extreme Explore and UCB with binary rewards. Thompson sampling performs the worst, but overall most decision rules are very similar. Binary rewards seem to make the training take more time, but the performance can still be as good. FedAvg reaches its highest performance 0.63 after 20 rounds.

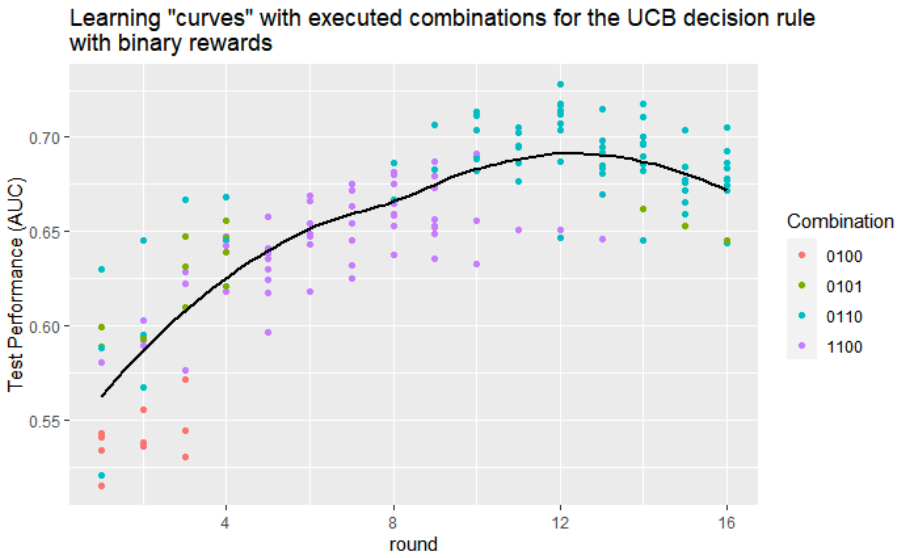

Figure 20: The learning curve and the executed actions for the UCB decision rule with reward type 9. The x-axis is the round of training and the y-axis is the test performance measured in AUC. The black line is the fitted spline based on 10 experiments. The colours show which combination was executed each round in all experiments. 0110 is the best combination and 1100 is the second to best.

this thesis should be able to do better. The price we pay for the ability to handle sparse data situations is that the MAB-FL model can only personalize for one worker per training session, as opposed to many other methods that personalize for every worker simultaneously during the same training session. Yet, personalising for every worker simultaneously may not be as efficient as it may seem, at least in situations where communication is expensive. This is because personalizing for all workers simultaneously is much more expensive in communication per round than in MAB-FL, as those models often are based on Federaded Averaging to some extent. Federated Averaging communicates with every worker (or a big proportion of them) every round, the MAB-FL model with pairs communicates with only two workers per round. Hence, only comparing the total number of rounds is not a fair comparison, as the cost of one round can differ much. The conclusion is that we often pay a price for personalizing all workers simultaneously as well and therefore personalizing for one worker at a time is not always a disadvantage.

We have found that how we measure performance increases is of utmost importance: one should compare the performance of the aggregated model ("the combination") with the one on the target worker. This reduces the dependencies between rewards and gives more stable and distinguishable estimated combination effects. Finding a suitable number of epochs per round is closely related to our ability to distinguish combination effects from each other. For us to observe a combination effect, in each round the workers in the combination must have been trained enough to deviate from the initial conditions of the given round, which is the global model of the previous round. Therefore one should set the number of epochs per round to be as big as possible. Increasing the number of epochs per round is also an efficient way of reducing the communication costs.

For the considered data set, adjusting the decision rules for non-stationary rewards was preferable in some settings but did not have a strong effect. This is likely to have to do with the short training time of about 10-15 rounds and that the best combination distinguishes itself already during the first few rounds. Also, as the training period is short, the decision rules own exploration incentive never fully disappears. However, the rewards showed the signs of non-stationarity and taking this into account may be important on other data sets where the training period is longer and the combination effects are more alike. That is the situation when we cannot trust the first impression. Extreme Explore, exponential recency weighting and discounted Thompson sampling can then be applied. One can also combine Extreme Explore with other decision rules by making Extreme Explore-rounds where one samples rewards from all reward distributions, or the relevant ones. One may schedule Extreme-Explore rounds or let each round be an Extreme Explore-round with some probability. We found that continuous rewards are mostly better than binary, but for UCB with binary rewards the performance was better than for UCB with continuous rewards. Thompson Sampling performed the worst, probably the rewards were too non-stationary for Thompson sampling. As compensating by adding a discount made the best

combination harder to distinguish, we conclude that Thompson Sampling is probably not the best decision rule for the MAB-FL model. There is little difference between the other decision rules, but that probably has to do with the data set and the short training period.

If one wants to use binary or discretized rewards, for example, to be able to conduct Thompson Sampling assuming a Beta prior and a Bernoulli likelihood, we need to have good threshold values. Trying out several different thresholds is time demanding and one might instead conduct a pre-training experiment with continuous rewards. For this one could use the model on the target worker or Federated Averaging and observe the quantiles of the increases of performance and base our thresholds on them. Another alternative is to, when training, save all performance increases and let the threshold be some quantile of the observed values. The same pre-training experiment can be used for all workers and it might give us good initial values for the global model. From the experiments we observe that the best threshold was surprisingly low. Allowing suboptimal combinations to pass the threshold sometimes is probably preferable to get zero-rewards, where we receive no information and the decision rule is not updated.

A challenge with this project has been to find an appropriate data set, one where the MAB-FL model is beneficial. That is, when the MAB-FL model performs better than both Federated Averaging and a model trained only on the target worker, the very biggest and smallest combination. To begin with, a suitable data set is one that splits naturally into different workers, such as cross-sectional data collected from different regions. Most available such data sets are apt for linear regression and are not complex enough for a neural network model. Furthermore, the size of those data set are often rather small with machine learning standards and after splitting the data set into different workers each worker ends up with very little data. Commonly used federated learning data sets tend to have too many workers to be used for the experiments in this thesis, as we end up with too many possible combinations to evaluate. In the next section we discuss how the approach considered in this thesis can possibly be combined with clustering analysis for it to be applicable in situations with many workers. This would mean that we extend the model from being a cross-silo federated learning model to a cross-device model. One can select a suitable amount of workers from an existing federated learning data set and disregard the others. However, it does not guarantee that the created sub-data set is suitable for evaluating the MAB-FL method. This is because the workers are often too different from each other for any combination to be beneficial and the best option is then to only use the target worker. The opposite situation also occurs, that the combinations become too similar for us to be able to distinguish them and Federated Averaging is then optimal. Evaluating a data set's suitability is time demanding as first a neural network model needs to be set up and tuned and then many time consuming experiments need to be run. Therefore we have refrained from a more extensive search of suitable data sets. Nevertheless, one should try the MAB-FL model on some more data sets in future research. It

would be interesting to see if we observe the same behaviours on different data sets, for example, the non-stationary rewards.

10 Future work

As mentioned in the report, the success of the MAB-FL model is depending on that we stop the training the global model in time and avoid going into overfitting. How to do this has not been investigated as the need to stop the training before the model becomes overfitted is a common problem and standard methods may apply. However, stopping in time for the MAB-FL model, should be a little more complicated than for a normal federated learning model. This is because dips and decreases in performance late in training happen more frequently, as we sometimes aggregate suboptimal combinations. One way to make stopping in time easier might be to train for fewer epochs per round when the increases in performance get smaller and in that way extend the period within which we should stop. It might also be possible to fit a cubic spline during training and stop when the slope turns negative. Another issue to consider is to always save the so far best model parameters and go back to those when we realize that we have gone into overfitting.

If one conducts a sequence of experiments, there might be a way to use the accumulated knowledge in the next experiment. One way to do so is to set initial values for the estimated expected rewards based on the last experiment. Another way of utilizing the accumulated knowledge may be to use the validation performances from the earlier experiments to fit a cubic spline and to stop the training when the slope of the spline turns negative. It may also be possible to use the information from the personalization of one worker for the personalization of another. For example, the combination effect may be somewhat symmetric: if the combination including the workers a and b is optimal when personalizing for worker a, it may also be beneficial when personalizing for worker b.

The results of the MAB-FL personalization approach might improve if one manages to aggregate the workers with more care and consideration. As of now, the models are given weights proportional to the size of the local data set. Averaging should be optimal when the data has the same distribution on all workers, but in federated learning, homogenous data is often too much to hope for. It might be a good idea to give a bigger weight to the target worker, for example 70% weight to the target worker and 30 % to the other worker/workers in the combination. This is reasonable as the data on the target worker should be the most suitable for its own test data, but the target worker model may need some influence from other workers to compensate for little data and avoid overfitting.

It might work to combine MAB-FL with other personalization techniques. These often proceed from Federated Averaging and this approach is merely an extension of Federated Averaging by averaging over the workers we find appropriate instead of all of them. The MAB-FL approach should also be possible to combine with cluster analysis. Clustering workers is probably hard as we cannot pool the data, but it still might be possible using some more sophisticated methods. Clustering model updates, however, is pretty simple and is done in, for example, Sattler *et al.* (2020) with good results. Workers with similar model updates are likely to have been trained on similarly distributed data and these workers might therefore complement each other. Yet, clustering does not remove the need to evaluate combinations entirely. The fact that two workers, or their parameter updates, end up in the same cluster does not guarantee that they complement each other in a way that affects the response variable. This is especially true in situations when we have many explanatory variables and have not performed dimensionality reduction or removed variables lacking correlation with the response variable. However, we will get subsets of workers which could potentially improve the models from each other. Cluster analysis may therefore be able to propose combinations with promise, which we then evaluate using the techniques of MAB-FL. Being able to reduce the number of combinations to consider is of great importance when there are a lot of combinations. This is because we have a limited time period to try out different combinations. The time limit is the time it takes to train the global model. Fewer combinations to consider increases our chances of evaluating them correctly and finding the optimal combination and thus the optimal personalized model. The similarity of the models updates from different workers might also be used to decide weights when aggregating.

In this thesis we always train for the same number of epochs per round for every round and worker. This is probably not optimal, especially when the amount of data on each worker differs and more epochs might be better early in the training, when the initial conditions are worse. This is because if the global model is poorly adapted to the local data more training is needed for it to reach a good fit. One might consider to train the local model on every worker for each round until the loss stabilizes or until we slightly overfit.

One might be able to extend the application of MAB-FL and use it for the opposite of personalization, i.e., to make a good general model. One could focus on which workers to leave out rather than include and in that way one might be able to eliminate poisonous workers or workers not contributing.

11 References

Arivazhagan, M. G. *et al.* (2019) 'Federated Learning with Personalization Layers' https://arxiv.org/pdf/1912.00818.pdf

Fallah, A., Mokhtari, A. and Ozdaglar, A. (2020) 'Personalized Federated Learning: A Meta-Learning Approach'. https://arxiv.org/pdf/2002.07948.pdf

Goodfellow, J., Bengio, Y. and Courville, A. (2016), 'Deep Learning', MIT Press.

Hand, D. J. and Till, R. J. (2001) 'A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems', Machine Learning, 45(2), p. 171. doi: 10.1023/a:1010920819831.

Hoeffding, W. (1963). 'Probability inequalities for sums of bounded random variables'. Journal of the American Statistical Association. 58 (301): 13–30.

Huang,J. and Ling, C. X. (2005) 'Using AUC and Accuracy in Evaluating Learning Algorithms', IEEE Transactions on Knowledge & Data Engineering, 17(3), pp. 299–310. doi: 10.1109/TKDE.2005.50.

Huigol, P. (2020) 'Bias and Variance in Machine Learning', Analytics Vidhya, https://www.analyticsvidhya.com/blog/2020/08/bias-and-variance-tradeoff-machine-learning/

Kairouz, P. *et al.* (2021) 'Advances and Open Problems in Federated Learning'. arXiv:1912.04977. https://arxiv.org/pdf/1912.04977.pdf.

McMahan, H.B. *et al.* (2016) 'Communication-Efficient Learning of Deep Networks from Decentralized Data'.Proceedings of the 20 th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017. https://arxiv.org/pdf/1602.05629.pdf

Raj, V. and Kalyani, S. (2017) 'Taming Non-stationary Bandits: A Bayesian Approach'. arXiv:1707.09727. https://arxiv.org/abs/1707.09727

Sattler, F., Müller, K., and Samek, W. (2020). 'Clustered Federated Learning: Model-Agnostic Distributed Multi-Task Optimization under Privacy Constraints'. IEEE transactions on neural networks and learning systems, PP.

Sutton, R. and Barto, A. (2018) Reinforcement Learning An Introduction. MIT Press, Second edition.

Wang, Hao, Kaplan, Zakhary, Niu, Di, & Li, Baochun. (2020). Optimizing Federated Learning on Non-IID Data with Reinforcement Learning. IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, 1698-1707.

12 Appendix

Weights of the exponential recency average sums to one

We here prove that we by updating the estimated mean as in Equation (2), i.e.

$$q^*_{t+1}(a) = q^*_t(a) + \frac{\mathbb{1}_{A_t=a}}{N_t(a)}\big(R_t - q^*_t(a)\big)$$

we still get weights summing to one.

Proof:

Recall that we can expand the above to

$$q^*_{t+1} = q^*_t + \alpha(R_t - q^*_t) = (1-\alpha)^t q^*_1 + \sum_{i=1}^{t} \alpha(1-\alpha)^{t-i} R_i.$$

Then

$$\begin{aligned}
(1-\alpha)^t + \sum_{i=1}^{t} \alpha(1-\alpha)^{t-i} &= (1-\alpha)^t + \alpha(1-\alpha)^t \sum_{i=1}^{t} \frac{1}{(1-\alpha)^i} && (13)\\
&= (1-\alpha)^t + \alpha(1-\alpha)^{t-1} \sum_{i=0}^{t-1} \frac{1}{(1-\alpha)^i} && (14)\\
&= (1-\alpha)^t + \alpha(1-\alpha)^{t-1} \left(\frac{1 - \left(\frac{1}{(1-\alpha)^t}\right)}{1 - \frac{1}{1-\alpha}} \right) && (15)\\
&= (1-\alpha)^t + \alpha(1-\alpha)^{t-1} \left(\frac{\frac{(1-\alpha)^t - 1}{(1-\alpha)^t}}{\frac{-\alpha}{1-\alpha}} \right) && (16)\\
&= (1-\alpha)^t + \alpha(1-\alpha)^{t-1} \left(\frac{(1-\alpha)^t - 1}{-\alpha(1-\alpha)^{t-1}} \right) && (17)\\
&= (1-\alpha)^t - ((1-\alpha)^t - 1) && (18)\\
&= 1 \quad \square && (19)
\end{aligned}$$

We have mainly utilized that we can rewrite the sum to a standard finite geometric sum for which we have a closed form expression, i.e. $\sum_{i=0}^{n} ar^i = a\frac{1-r^{n+1}}{1-r}$. In

our case $a = 1$ and $r = \frac{1}{1-\alpha}$ as $\frac{1}{(1-\alpha)^i} = \left(\frac{1}{1-\alpha}\right)^i$.

The Beta distribution is a conjugate prior for the Bernoulli likelihood function

We are here to show the claim of Example (4.1), the posterior distribution when having a Bernoulli likelihood and a Beta prior.

Let R be Bernoulli distributed with parameter ϕ and ϕ be Beta distributed with parameters α and β, i.e.

$$p(r|\phi) = \phi^r(1-\phi)^{1-r},$$

$$p(\phi) = \frac{\phi^{\alpha-1}(1-\phi)^{\beta-1}}{B(\alpha,\beta)}$$

where $B(\alpha, \beta)$ is the Beta function, a normalizing constant given that α and β are constants.

We now show that if we have a Bernoulli likelihood and a Beta prior distribution we again get a Beta posterior distribution. We show this by applying Bayes formula:

$$p(\phi|r) = \frac{p(r|\phi)p(\phi)}{p(r)} \propto p(r|\phi)p(\phi)$$

We treat $p(r)$ as a constant as it contains no random component, it is merely a constant with a value between zero and one.

Now,

$$\begin{aligned} p(\phi|r) &\propto p(r|\phi)p(\phi) && (20)\\ &= \phi^r(1-\phi)^{1-r}\frac{\phi^{\alpha-1}(1-\phi)^{\beta-1}}{B(\alpha,\beta)} && (21)\\ &\propto \phi^r(1-\phi)^{1-r}\phi^{\alpha-1}(1-\phi)^{\beta-1} && (22)\\ &= \phi^{r+\alpha-1}(1-\alpha)^{\beta-r+1-1} && (23) \end{aligned}$$

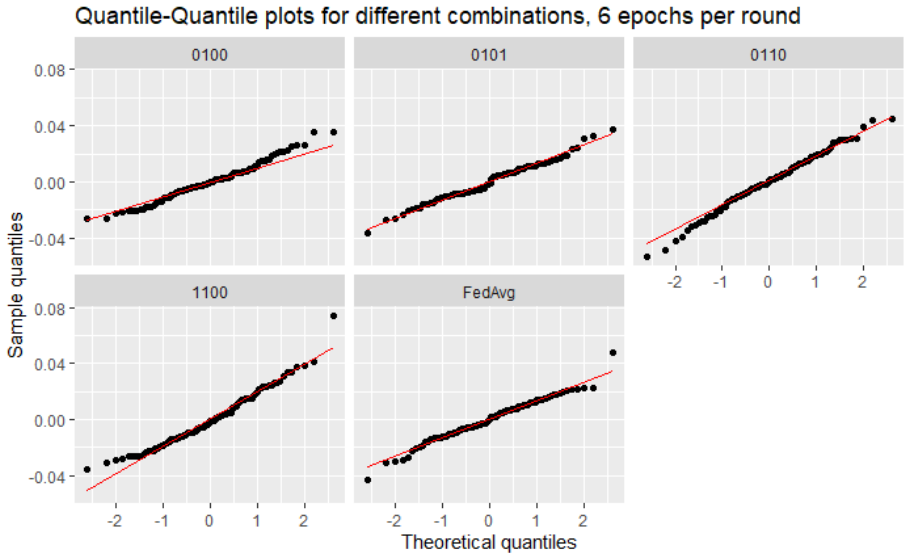

Figure 21: Complementary plot to section 8.2.1. Quantile-Quantile plots for the residuals when training with 6 epochs per round. It is reasonable to make a normal distribution approximation for the residuals.

which is proportional to a $Beta(\alpha+r, \beta+1-r)$ density function. This means the normalizing constant equals $B(\alpha+r, \beta+1-r)$, and the posterior distribution is $Beta(\alpha+r, \beta+1-r)$.

Complementary Figures

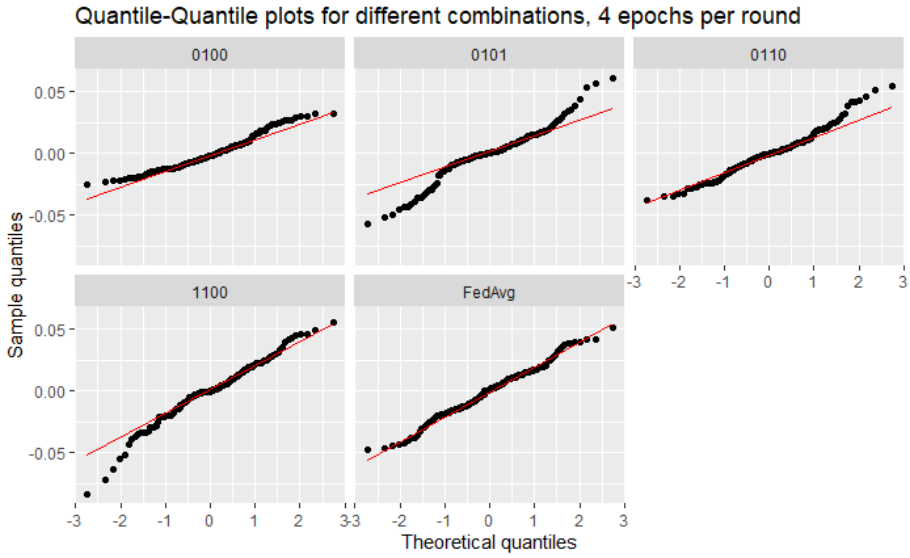

Figure 22: Complementary plot to section 8.2.1. Quantile-Quantile plots for the residuals when training with 4 epochs per round. It is not reasonable to make a normal distribution approximation for the residuals.

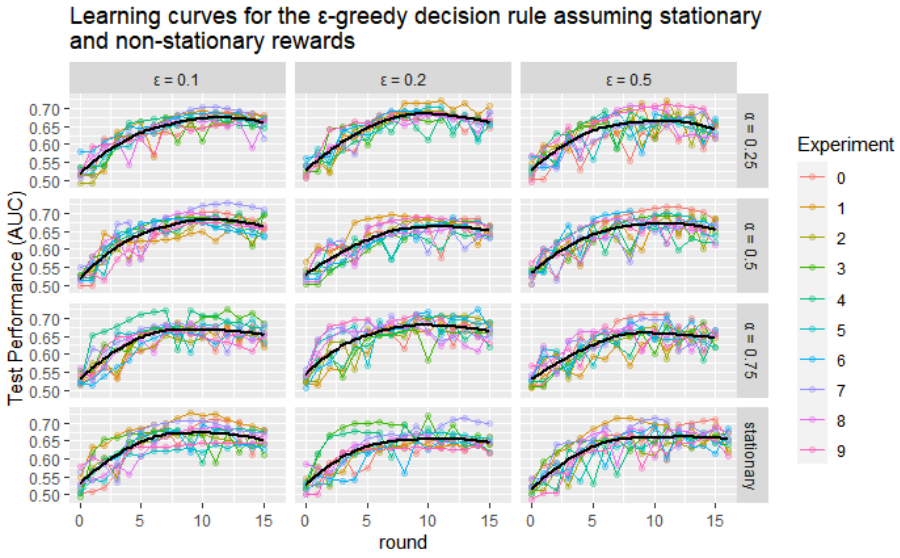

Figure 23: Learning curves for the ε-greedy decision rule with 10 experiments for each setting. The black lines are fitted cubic splines. The y-axis measures the performance measured in AUC on the test set of the target worker. The x-axis gives the round of training. The columns are different values for ε. For the interpretation of α andε see Figure 16 "Stationary" means that we use the ordinary weighted mean. With higher ε the performances oscillate more, yielding more variance. For $\varepsilon = 0.1$ and $\varepsilon = 0.2$ we see reduced variance as α becomes smaller. Other than that there are no obvious patterns.

0.750.75

Figure 24: The estimations of the expected rewards throughout training for 10 experiments per setting with the ε-greedy decision rule. The y-axis is the estimated expected reward and the x-axis is the round of training. Each column is a different value of the parameter ε and each row is a different value of α. "Stationary" means that we assume stationary rewards and have no α-parameter. For interpretations of ε and α see the caption on Figure 23. Each colour is a combination and the lines are fitted cubic splines. All estimates has 0.05 as initial values, which is quite optimistic, but judging by some figures not optimistic enough. High initial values makes sure that each combination is evaluated at least once. We know from Section 8.2.1 that 0110 is the best combination. For stationary rewards, the bottom row, the estimates vary the most between experiments. For $\varepsilon = 0.05$ and assuming stationary rewards the best combination cannot be distinguished to be 0110. For $\alpha = 0.25$ the estimates are all very close. $\alpha = 0.5$ or $\alpha = 0.75$ seems to make the combinations the most distinguishable, the values of ε seem to be of little importance.

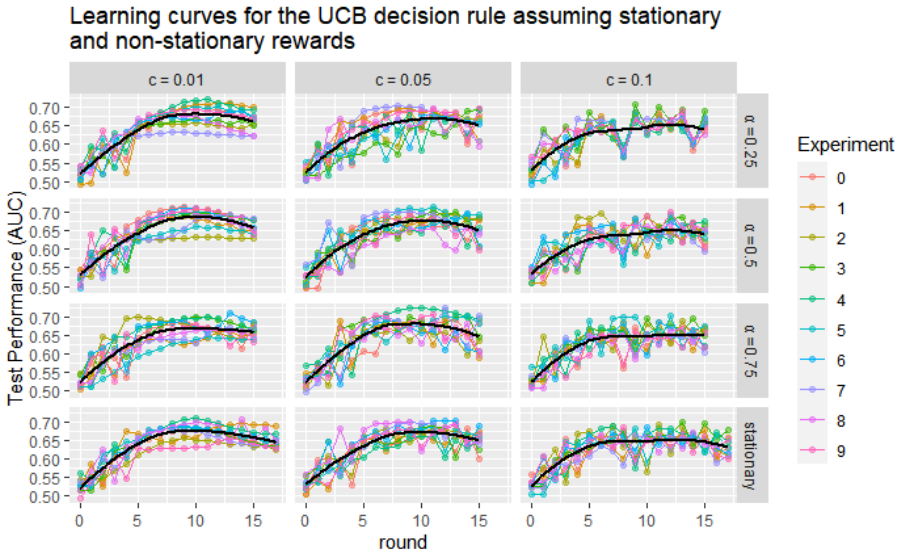

Figure 25: Learning curves for the UCB decision rule with 10 experiments for each setting. The black lines are fitted cubic splines. The y-axis measures the performance measured in AUC on the test set of the target worker. The x-axis gives the round of training. For interpretations of c and α see the caption of Figure 25. With higher c the performances oscillate more, yielding more variance. With $c = 0.01$ we can see that the oscillation stops after 5 rounds. Other than that there are no obvious patterns, the behaviour seems the same when we assume stationary and non-stationary rewards.

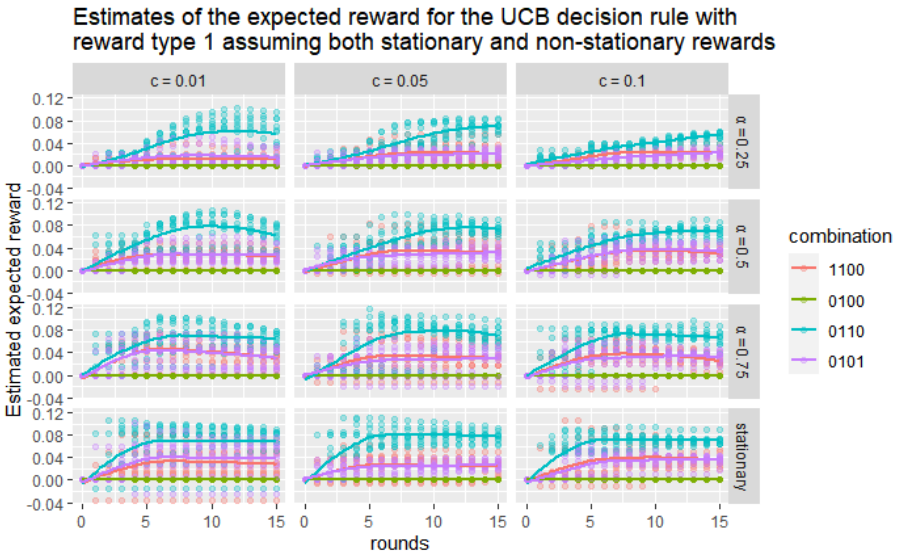

Figure 26: The estimations of the expected rewards throughout training for 10 experiments per setting with the UCB decision rule. The y-axis is the estimated expected reward and the x-axis is the round of training. Each column is a different value of the parameter c and each row is a different value of α. "Stationary" means that we assume stationary rewards and have no α-parameter. For interpretations of c and α see the caption on Figure 25. Each colour is a combination and the lines are fitted cubic splines. We know from Section 8.2.1 that 0110 is the best combination. For $\alpha = 0.25$ the effect of increasing values of c seems to be that the combination effects take longer time to distinguish. $\alpha = 0.75$ or assuming stationary rewards seems to be optimal. There the estimates also seem to converge after round 5.

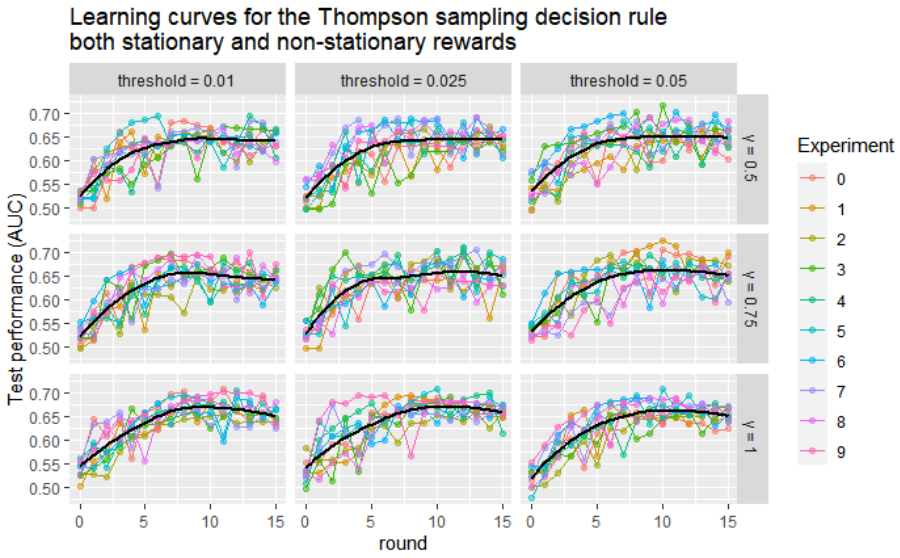

Figure 27: Learning curves for the Thompson Sampling decision rule with 10 experiments for each setting. The black lines are fitted cubic splines. The y-axis measures the performance measured in AUC on the test set of the target worker. The x-axis gives the round of training. For interpretations of the threshold and γ see the caption of Figure 27. $\gamma = 1$, or ordinary Thompson sampling, seems to give less variance, other than that there are no obvious patterns.

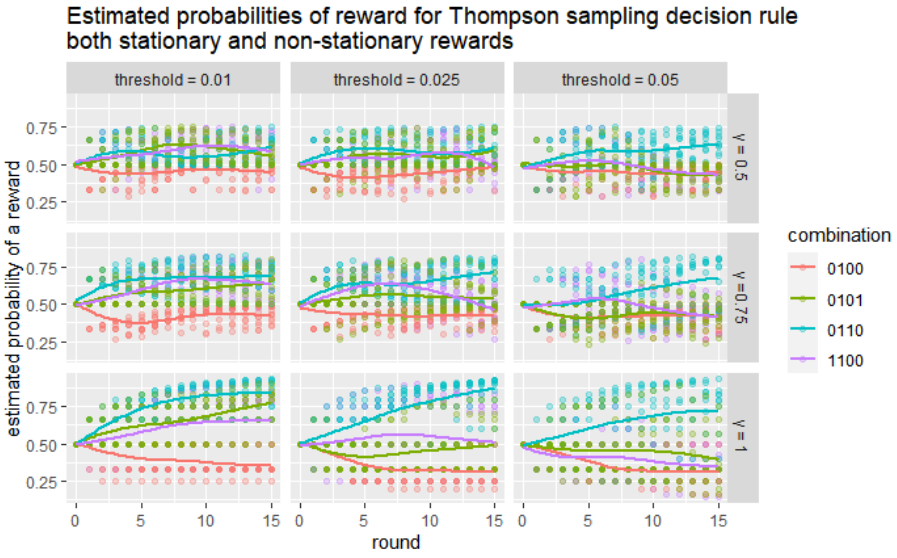

Figure 28: The estimations of the expected rewards throughout training for 10 experiments per setting with the Thompson Sampling/Discounted Thompson Sampling decision rule. The y-axis is the estimated expected reward and the x-axis is the round of training. Each column is a different value of reward threshold and each row is a different value of γ. For interpretations of the threshold and γ see the caption on Figure 27. Each colour is a combination and the lines are fitted cubic splines. We know from Section 8.2.1 that 0110 is the best combination. For the threshold value 0.05 the combination (0110) can easily be distinguished as the best combination. The same goes for the combination of the threshold 0.025 and $\gamma = 1$, but for the other parameter pairs the estimated effects are more intertwined. A high threshold and a high value of γ seems to be important for us to be able to distinguish the combinations.