Transfer learning for hierarchical forecasting: Reducing computational efforts of M5 winning methods*

Arnoud P. Wellens[a,*], Maxi Udenio[a], Robert N. Boute[a,b]

[a]*KU Leuven, Research Center for Operations Management, Naamsestraat 69, Leuven, 3000, Belgium*
[b]*Vlerick Business School, Technology and Operations Management Area, Reep 1, Ghent, B-9000, Belgium*

Abstract

The winning machine learning methods of the M5 Accuracy competition demonstrated high levels of forecast accuracy compared to the top-performing benchmarks in the history of the M-competitions. Yet, large-scale adoption is hampered due to the significant computational requirements to model, tune, and train these state-of-the-art algorithms. To overcome this major issue, we discuss the potential of transfer learning (TL) to reduce the computational effort in hierarchical forecasting and provide proof of concept that TL can be applied on M5 top-performing methods. We demonstrate our easy-to-use TL framework on the recursive store level LightGBM models of the M5 winning method and attain similar levels of forecast accuracy with roughly 25% less training time. Our findings provide evidence for a novel application of TL to facilitate practical applicability of the M5 winning methods in large-scale settings with hierarchically structured data.

Keywords: M5 Accuracy Competition, Computational Requirements, Transfer Learning, LightGBM, Hierarchical Forecasting

1. The rise of machine learning in recent M-competitions

For a long time, simple statistical models such as exponential smoothing dominated the field of forecasting and were the default in commercial software packages (Fildes et al., 2019). These univariate models are easy to compute and were historically able to achieve similar levels of forecast accuracy to more complex methods (Makridakis et al., 2020a). This observation from the first forecasting competition by Makridakis and Hibon (1979) was reconfirmed in the first three M-competitions (Makridakis et al., 1982, 1993; Makridakis and Hibon, 2000). The M3 winning model, for instance, combined linear regression with simple exponential smoothing with drift (Assimakopoulos and Nikolopoulos, 2000). Although simple by design, it outperformed the more sophisticated methods in the M3 competition as well as every model in the NN3 competition, which was organized nearly a decade later to promote the use of machine learning (ML) for forecasting (Crone et al., 2011; Hyndman, 2020).

This changed in 2015, when ML methods started to dominate multiple large-scale forecasting competitions (e.g., the Rossmann Store Sales competition,[2] the Wikipedia Web Traffic Time Series Forecasting competition,[3] the Corporación Favorita Grocery Sales Forecasting competition,[4] etc.). Bojer and Meldgaard (2021) attribute this sudden rise of ML to multiple innovations in

*Corresponding author
Email address: `arnoud.wellens@kuleuven.be` (Arnoud P. Wellens)
To appear in the *International Journal of Forecasting*
[2]https://www.kaggle.com/c/rossmann-store-sales
[3]https://www.kaggle.com/c/web-traffic-time-series-forecasting
[4]https://www.kaggle.com/c/favorita-grocery-sales-forecasting

the field, such as embedded layers (Guo and Berkhahn, 2016), long short-term memory models (Hochreiter and Schmidhuber, 1997) and the launch of well-performing Gradient Boosting Decision Tree (GBDT) methods (Chen and Guestrin, 2016; Ke et al., 2017). The latter build a large set of decision trees sequentially, so that each consecutive decision tree improves upon the errors made by the previous ones (Friedman, 2001; Ke et al., 2017).

The M4 (early 2018) was the first M-competition where sophisticated methods substantially outperformed simpler statistical models by combining ML with statistical features (Makridakis et al., 2018a, 2020a). The top-performing methods of M4, however, are rather complex to implement (Makridakis et al., 2020b). Simpler and more straightforward ML implementations that are easier to model and tune in turn performed rather poorly in M4. This was different in the most recent M5 Accuracy competition, when relatively straightforward ML models improved the forecast accuracy of the benchmarks up to roughly 20% (Makridakis et al., 2020b). It was the first M-competition where pure ML methods achieved superior results compared to simple statistical and more sophisticated methods. The winning method produced sales forecasts using an equal weighted combination (ensemble) of pure Light Gradient Boosting Machine (LightGBM) models, which is an implementation of GBDT. It is striking that almost all of the M5 top 50 performing methods made use of LightGBM (only the method at the third place was solely based on neural networks, also a ML-based method). This is likely because LightGBM models perform well without extensive data preprocessing or hyperparameter tuning and are easy to use. This is highlighted by the fact that an undergraduate student with little experience in the field of forecasting won the M5. In addition, LightGBM models can easily exploit cross-series information by learning across different time series. 45 of the top 50 performing methods trained LightGBM models per group of time series that belong to the same store/category/department. This is known as cross-learning and has shown to be effective in the M4, the M5 and other forecasting competitions (Makridakis et al., 2020b; Bojer and Meldgaard, 2021). All these characteristics make LightGBM a well-performing and easy-to-adopt forecasting method for practical purposes.

Despite their ease of use, these top-performing LightGBM methods come with significantly more computational requirements to model, tune and train the algorithms compared to commonly used statistical methods (Makridakis et al., 2020a, 2018b). The computational burden mainly stems from the fact that these methods need to train multiple ML models for each store, category or department from scratch, even though evident similarities such as seasonality or promotional effects exist throughout the hierarchical structure of the time series – especially between closely related time series. We believe this high computational burden may hinder the adoption of these ML methods in retail settings with thousands of products and dozens of stores (Seaman, 2018). In this article we therefore introduce the potential of transfer learning (TL) to reduce the computational requirements in hierarchical forecasting and propose an easy-to-use TL framework to reduce the training time of the M5 top-performing models. Based on the hierarchical structure of the dataset, our TL framework trains a handful of general ML models from scratch, and uses TL to reuse model parameters of pretrained ML models for similar stores, categories or departments to train the tree-based models more efficiently. We illustrate this idea on the recursive store level LightGBM models of the M5 winning method and attain similar levels of forecast accuracy with roughly 25% less training time. We also discuss a number of limitations of the TL framework and propose further research to address these. With our work we aim to demonstrate how TL can facilitate the applicability of the M5 top-performing models in settings with hierarchically structured data.

2. Forecast accuracy versus computational requirements

Before the advent of ML, the computational requirements of sales forecasting received little attention due to the longstanding belief that simple statistical forecasting models could perform equally well as more sophisticated methods (Makridakis et al., 2020b) and the improved accessibility of computational power (Waldrop, 2016). With recent advances in ML, there is evidence that 'simple' ML methods can outperform statistical models. However, substantial computational power is required to achieve this high level of forecast accuracy in large-scale settings.

Makridakis et al. (2018b) compare eight statistical methods with eight families of popular ML methods on the M3 dataset. Not surprisingly, they find that to achieve the same level of forecast accuracy, ML methods require significantly more computational requirements in terms of training and prediction time than traditional statistical methods. Spiliotis et al. (2020) make a similar comparison and include different ML methods like GBDT and more sophisticated statistical methods to forecast the daily sales demand of a major retail company in Greece. They report that ML requires, on average, 4.25 times longer computation times than statistical methods. Note, however, that these results refer to ML methods using local models, i.e., methods that train one model per time series. Recently, globally trained ML models, which train one model per group of time series, have also started to achieve competitive results (Smyl, 2020; Januschowski et al., 2020; Bandara et al., 2020). Global models have the advantage that they require fewer models overall, enabling a potential reduction in computational requirements (Makridakis et al., 2020b). Spiliotis et al. (2020) showcase a set of global ML models with only one third of the computational requirements of their statistical benchmarks. This, however, cannot be generalized to *all* global models, especially not to the global methods that won the most recent forecasting competitions (Bojer and Meldgaard, 2021). The top-three global methods of the M4 competition, for example, required on average 10 to 35 times more computation time than popular benchmarks such as ETS and autoARIMA (Makridakis et al., 2020a).

The computational requirements of the top-performing LightGBM methods in the M5 Accuracy competition cannot be neglected either. These methods train multiple ML models with thousands of decision trees and a wide variety of input features (e.g., past sales data, explanatory variables and rolling statistics). For example, the M5 global random forest benchmark, which trains a single model with 500 decision trees (Makridakis et al., 2020b) requires only a fraction of the total computation time of the 660,000 decision trees that are trained under the M5 winning method. The popularity of these computationally demanding methods can, most likely, be attributed to the excellent performance of a particular LightGBM method, whose code was shared during the M5, using a Kaggle notebook.[5] The latter became one of the most examined and influential LightGBM methods of M5. It is an implementation of the Recursive Store Level LightGBM model (RSLLM) that groups the time series of the product-store level per store and trains one pure LightGBM model per store. It is called recursive as the models use their own, previously made, sales forecasts as input features to make prediction for later timestamps. The M5 winning method used this notebook (with minor adjustments) in an ensemble of five other implementations of this notebook by adjusting the hyperparameters, the (recursive) features and the group of time series each model was trained on. As in most submissions, these groups of time series were organized using the hierarchical structure of the time series to enable cross-learning (Makridakis et al., 2020b). The winning method trained two models per store, two per store-category and two per store-department,

[5]https://www.kaggle.com/kyakovlev/m5-three-shades-of-dark-darker-magic

resulting in an ensemble of six pure LightGBM models per forecast. This required training 220 different LightGBM models to cover the 10 stores, 30 store-categories, and 70 store-departments of the M5 limited dataset, which includes 3,049 different products.

It is worth noting that before training any definitive model, an experimental phase takes place where different model architectures (i.e., ensembles, recursive vs. non-recursive forecasting, ...), input features and hyperparameters are tested. The computational cost of this modeling and tuning step largely depends on the total training time of the forecasting models as it iteratively retrains the models with small differences in model architecture, features and hyperparameters (Januschowski et al., 2020). Using different hyperparameters per model is often beneficial (and was empirically tested on the M5 dataset),[6] but is computationally expensive. Remark that the winning method (which did not tune most of the hyperparameters, as the 210 LightGBM models share the exact same hyperparameters) already requires substantial training times.

The high computational requirements to model, tune and train different forecasting models is especially relevant for large-scale retail applications with thousands of products and dozens of stores. Scaling their forecast methods is one of the major forecasting challenges mentioned by retailers due to the computational requirements (Fildes et al., 2019). Walmart, for example, needs to make over a billion unique forecasts a day to stay operational in the US alone. This number even exceeds one trillion if unique sales forecasts per product and per region of Walmart's online marketplace are included in the calculation (Seaman, 2018). Scaling the M5 winning method to Walmart's 10,500 stores[7] would result in 231,000 LightGBM models (excl. their online sales forecasts, any long term forecasts and the fact that Walmart may offer over 200,000 products in one store instead of only 3,049). To put into perspective, training one RSLLM of the winning method on 3,049 time series takes roughly 10-20 minutes using a state-of-the-art €11K server (without GPU). In addition to that, most retailers retrain their models from scratch when new data or products come in to ensure the highest possible level of forecast accuracy. This raises the question whether the incremental benefit in forecast accuracy justifies the additional costs of these computational expensive forecasting models (Makridakis et al., 2020b).

3. Transfer learning to reduce computational efforts

We study how transfer learning (TL) can reduce the computational burden of training the M5 top-performing ML methods without compromising on forecast performance. We briefly introduce the concept of TL in this section. Section 4 formally describes our TL framework that can be used when data is organized in a hierarchical structure. Section 5 illustrates how TL speeds up training time in the family of LightGBM models that is used by most M5 top-performing methods.

At its most basic, TL facilitates the learning of a new task (i.e., a target task) by utilizing knowledge from another task that has already been explored (i.e., a source task; Torrey and Shavlik, 2010). TL is an intuitive concept that comes naturally to us, humans, when we need to learn a new task. A tennis player, for example, who is learning how to play ping-pong will automatically exploit their expertise in tennis (the source) to speed up their training in ping-pong (the target). In ML, however, TL does not come naturally. It is an explicit design decision that is used to transfer parameters from pretrained *source models* to untrained *target models* that exhibit similarities in their structure. Such an approach is known as the parameter-transfer approach

[6]https://www.kaggle.com/kyakovlev/m5-three-shades-of-dark-darker-magic
[7]https://corporate.walmart.com/our-story/our-locations (accessed 15 July 2021)

(Pan and Yang, 2009), and is typically used when data collection is impossible or expensive (Afrin et al., 2018; Taylor and Stone, 2009; Fang et al., 2019). Karb et al. (2020), for instance, use TL to produce ML sales forecasts for a newly launched product that lacks the historical data required for training. They pretrain a neural network on similar products for which they have sufficient data and use this pretrained model as a starting point to forecast new product sales. TL is also useful when moving or collecting certain data is prohibited by privacy barriers or regulations, such as the General Data Protection Regulation (GDPR). By transferring the pretrained models, insights from valuable data sources can be reused without exposing the raw data itself (Hirt et al., 2020).

Transfer learning is also used in reinforcement learning, which requires multiple episodes (such as simulation or real-life experiments) that can be very expensive due to the sheer volume needed. For example, training AlphaGo – the algorithm famous for being the first to beat the best GO player in the world – required simulating over 30 million games, which at the time (early 2016) cost roughly 25 million dollars in computation power (Silver et al., 2016; Gibney, 2017). TL thus shows promise to reduce the amount of episodes by reusing knowledge from pretrained models on related problems (Taylor and Stone, 2009; Finn et al., 2017).

The computational burden of the M5-winning LightGBM methods stems from the large number of models that are trained from scratch, i.e., multiple models for each store, category, or department. Given the natural similarity between hierarchically structured time series, we believe that not all models need to be fully trained. Product sales in retail are often driven by similar input features such as promotions, holidays, weather forecasts and more, and thus may be correlated. Mukherjee et al. (2018) use a similar idea to forecast tens-of-thousands of products from Flipkart, India's largest e-retailer. To cope with the scale and diversity of the time series, they start by training a single global neural network over all time series. They then transfer the (trained) model parameters to each category of (relatively similar) products, and further train the model parameters per category, which we call fine-tuning. Although the reduction in computational requirements is not examined, this should require less training time than training all models from scratch. Thus far, little is known as to how this approach can be extended to GBDT methods or how TL impacts the forecast accuracy. A number of papers have shown that a suboptimal selection of hyperparameters (Nikolopoulos and Petropoulos, 2018) or forecasting models (Petropoulos and Grushka-Cockayne, 2021; Ashouri et al., 2019) have only a negligible effect on the forecast accuracy in an out-of-sample test. As a result, reusing model parameters of related forecasting models, even if they are not completely similar, seems to be a viable approach without leading to a substantial accuracy loss.

To reduce the computational burden of LightGBM methods, we propose a TL framework that trains a subset of representative *source models* and transfers the (trained) model parameters to the *target models*, which are then fine-tuned rather than fully trained from scratch. Given that model training is a major bottleneck (re. computational requirements) of LightGBM implementations, a reduction in the number of models that need to be trained will result in training time reductions. For the ease of the reader, from here on the term 'training a model' refers to fully training a model from scratch without transferring any model parameters. Remark that model parameters are in sharp contrast to hyperparameters as the hyperparameters are typically set manually during modeling, while model parameters are automatically estimated during training. To the best of our knowledge, our study is the first to apply TL on GBDT methods with the objective to speed up training time while achieving similar levels of forecast accuracy.

4. Transfer Learning framework

4.1. Preliminary matters

To formalize our TL framework, we introduce the following notation. Let L_i, $i \in [1, m]$ be the i^{th} level of a hierarchical data structure with a total of m levels, where L_1 is the highest and most aggregated level (such as the total unit sales), L_2 the first level of disaggregation (e.g., sales per state), and L_m the lowest and most disaggregated level (e.g., sales per product-store level). At its lowest level, the data structure contains n_m time series, which – following the hierarchical structure – can be aggregated into n_i time series at each aggregation level L_i. Note that $n_i \geq n_{i-1}$; i.e., the number of time series generally decreases in the level of aggregation. We define a node as any aggregated time series in the hierarchy. When two or more nodes of level L_i are contained by the same node at L_{i-1}, the latter is described as the parent of the former. In general, nodes that are contained by the same node at any given (higher) level, are said to share a common *ancestor*.

In absence of any data grouping, producing forecasts for the n_i time series at level L_i requires training one LightGBM model for each time series. Most LightGBM implementations exploit cross-series information by learning across groups of time series. This way, one LightGBM model is trained per *group* of time series. These groups of related time series are naturally found by grouping the time series based on common ancestor nodes. We denote g, with $g \in [0, i]$, the level upon which the time series are grouped, relative to the level of the forecasted time series, L_i. Here, $g = 0$ indicates no grouping (i.e., one LightGBM model per time series), $g = 1$ indicates grouping per parent node, $g = 2$ grouping per grand-parent node, etc. By grouping the time series at level g, producing forecasts for the n_i time series of L_i, requires training one LightGBM model for each of the n_{i-g} nodes of level L_{i-g}. Thus, the number of LightGBM models required to produce the forecasts for the time series of L_i decreases in g. A conclusive answer to determine the optimal value of g is yet to be found as different grouping strategies have successfully been applied on the M5 and other large-scale datasets (Makridakis et al., 2020b; Mukherjee et al., 2018; Bojer and Meldgaard, 2021). Choosing higher grouping levels benefits cross-learning and typically reduces the overall computational requirements (Makridakis et al., 2020b). Nevertheless, pooling more time series increases the model complexity as it often results in the inclusion of more input features, model parameters, and training data, and potentially deteriorates the forecast accuracy by including highly diverse time series within the same model (Mukherjee et al., 2018).

4.2. TL approach

Our TL framework is based upon a similar concept of grouping time series. However, rather than training one model per group of time series with grouping level, g, we define a higher-level grouping, g', which we use in conjunction with the lower level grouping, g. Specifically, we create one *source model* per grouping g' and one *target model* per grouping g. By only training source models from scratch, and copying these trained model parameters to the target models, our methodology effectively increases the grouping level. Yet, as we discuss next, we do so while keeping the number of features, model parameters and training data constant.

Denote $\mathcal{T}$ as the target models that group time series of L_i with grouping level g at level L_{i-g}, and effectively produce forecasts at L_i. The models $\mathcal{T}$ are the models that our TL framework wants to obtain with less computational efforts. While traditional GBDT methods would fully train n_{i-g} models, our TL framework, instead, performs a higher level grouping, where we group the time series at level $g' > g$ with common ancestors at level $L_{i-g'}$. The source models $\mathcal{G}$ are trained for each grouping g'. Once models $\mathcal{G}$ are trained at grouping level g', the model parameters are copied to all target models $\mathcal{T}$ at grouping level g, which are further fine-tuned without the need for wholesale training. As such, only $n_{i-g'}$ models are fully trained, whereas n_{i-g} models are fine-tuned. To minimize the total computational requirements of our TL framework, all the model parameters from the trained source models $\mathcal{G}$ are transferred to the target models $\mathcal{T}$ and no new features or new model parameters are introduced when fine-tuning these models. As a result, all input features and hyperparameters that are used to train the source models $\mathcal{G}$ will also be used by the target models $\mathcal{T}$. As we primarily focus on the forecast accuracy of the target models $\mathcal{T}$, we need to make sure that the target models $\mathcal{T}$ (with grouping level g) use the optimized features and hyperparameters for grouping level g. That means that we also train the source models $\mathcal{G}$ with the optimized features and hyperparameters for grouping level g, even though these models group at level g'. In addition to that, as the source models $\mathcal{G}$ use the same optimized hyperparameters of grouping level g, they should also be trained on a similar sized dataset of grouping level g. This is because the optimized hyperparameters of GBDT methods depend on the size of the training dataset (e.g., min_data_in_leaf for regularization). Given that $n_{i-g} \geq n_{i-g'}$, we therefore select a subset of the available time series of level g'. This can easily be achieved, for example, by sampling random training data of level L_i at the higher level grouping g'.

Note that our TL framework does not increase the training data per model, nor the number of features or hyperparameters, compared to traditional pooling approaches. As a result, it trains $n_{i-g'}$ similar-sized models instead of n_{i-g}, thus reducing the computational requirements by a factor of $n_{i-g}/n_{i-g'}$. Without further fine-tuning, this may come at the expense of a reduced forecast accuracy due to having dropped data or using the optimized features and hyperparameters for grouping level g instead of grouping level g'. For instance, applied to the M5 data, if g defines a grouping of time series at store level and g' at state level, then we define one source model $\mathcal{G}$ per state, which we train only with features available at store level. As store characteristics (e.g., store ID, store location, etc.) are a fixed value for models that only pool time series at the store level, store characteristics are input feature without any variance and are thus redundant. Therefore, these features are not included at grouping level g, and thereby not included at grouping level g' to train $\mathcal{G}$. As a result of this, the source models $\mathcal{G}$, which are trained per state, cannot differentiate the sales from one store to the sales in another store, and fail, e.g., to learn specific patterns that only apply to one store.

To alleviate this potentially reduced forecast accuracy, we further fine-tune each target model $\mathcal{T}$ by adjusting a subset of the pretrained model parameters. We therefore only adjust these prespecified parameters using the time series of level L_i which share the same ancestor at grouping level g while keeping the remaining pretrained model parameters fixed. Different fine-tuning strategies have been proposed for tree-based models, such that the adjusted models are capable of learning the specific patterns at grouping level g (Segev et al., 2016; Fang et al., 2019; Son et al., 2015). Two important parameters in tree-based forecasting models are the splitting criteria to classify the data based on feature values (defining the nodes of the decision tree), and the prediction value of the data points in each class (constituting the leaves in the decision tree). As optimizing the nodes of decision trees is the most time-consuming operation of training tree-based methods (Ke et al., 2017), we adjust the latter.

Since fine-tuning only adjusts a subset of the model parameters, this step requires less training data compared to the data that would be needed to train all model parameters entirely. The reason for this is that fine-tuning is less likely to overfit the training data. This results in efficient training times as the computational requirements depend on the amount of training data used and the number of parameters that are fine-tuned.

The total computational requirements to train the TL framework largely depend on the hierarchical structure of the time series and the need for fine-tuning the target models. In the case of the M5 dataset, for example, we have ten stores and three states. If g is the store level (with $n_{i-g} = 10$ stores) and g' the state level (with $n_{i-g'} = 3$ states), then, without taking the computational cost of fine-tuning into account, we reduce the total computational requirements with a factor of $n_{i-g}/n_{i-g'} = 10/3 = 3.33$ (i.e., 70% reduction in training time). The TL framework can reach even higher reductions in training time, when there are for example more stores per state. We do note that the fine-tuning step also requires computation time. However, by only fine-tuning the predictions of the leaves of the decision trees using few training data, we can still adjust the target models $\mathcal{T}$ to learn specific patterns, without recalculating the most time-consuming operations, namely the nodes of the decision trees. This should allow similar levels of forecast accuracy, while still benefiting from a substantial reduction in training time.

Remark, however, that the training time is only one component of the total computational cost of implementing forecasting methods. The two other components are: (1) the experimental phase to find adequate models and (2) the prediction phase to produce the forecasts (Januschowski et al., 2020). Our TL framework focuses on reducing the total training time while keeping the computational cost of (1) and (2) the same. The computational cost of (2) remains similar as the target models produced by the TL framework are identical in the model architecture, input features and hyperparameters compared to the forecasting models we want to replicate. As a consequence, producing forecasts with similar-sized GBDT models take equally long. The computational requirements during (1), however, may vary. The computational cost of (1) largely depends on the total number of models with different model architectures, features and hyperparameters that need to be trained and compared. Our TL framework keeps the number of model architectures and input features constant, but may slightly increase the total number of hyperparameters. Depending on the fine-tuning strategy, it may require an additional hyperparameter that needs to be optimized. As we show later in our numerical illustration, this additional hyperparameter can be optimized independently of the other model variations, which significantly reduces the total number of model variations.[8] Moreover, if the number of hyperparameters stays the same, the lower training time of the TL framework even decreases the computational requirements of (1) as it improves the speed of retraining models with different model architectures, features and hyperparameters. Finally, note that the experimental phase is typically a one-time investment, while retraining GBDT methods happens periodically when new data or products come in.

5. Numerical illustration

We illustrate the effectiveness of the TL framework on the Recursive Store Level LightGBM models (RSLLMs) of the M5-winning method. We do so by replicating this RSLLMs implementation with our TL framework. Section 5.2 compares the forecast accuracy and the training time of these RSLLMs against the application of our TL framework.

[8]As a numerical illustration: optimizing 10 hyperparameters with 10 different values dependent of each other gives a total of 10^{10} model variations, but only 100 variations when optimized independently.

5.1. Training the RSLLMs and our TL framework

The RSLLMs are trained using the code files of the M5 winning method,[9] and the M5 publicly available dataset.[10] Following the data structure of the M5 dataset, the most disaggregated level of the dataset is level L_{12}, where n_{12} includes $30,490$ different time series of (at most) 1,941 days of sales (roughly 5.3 years). The RSLLMs group the time series at the product-store level L_{12} per common ancestor at the store level L_3, such that the grouping level $g = 9$. We train one LightGBM model per store or ten LightGBM models in total as $n_{12-9} = 10$. These RSLLMs then produce sales forecasts at the product-store level L_{12}, which can be aggregated to obtain sales forecasts for every level of the hierarchy.

Our TL framework tries to replicate these RSLLMs (given their optimized input features, hyperparameters, grouping level g, etc.) with reduced computational requirements by grouping the time series of level L_{12} at a higher level g', using the hierarchy in the data. We set $g' = 10$, i.e., we group at the level of the state, and train one source model $\mathcal{G}$ per state, using the same code, features and hyperparameters of the RSLLMs. This results in $n_{12-10} = 3$ source models $\mathcal{G}$, i.e., one per state. As the hyperparameters of the RSLLMs are optimized for the average amount of training data per store, and we reuse these same hyperparameters to train the RSLLMs per state, we randomly sample training data such that the amount of training data sampled per state is equal to the amount of training data per store. Note that the total number of training instances per store in the M5 dataset is different as not all products are introduced at the same time in each store. By simply taking the average amount of training data per store in each state, we successfully eliminate the three-to-four-fold increase in training data (as each state covers three to four stores).

We could stop here, and use the trained source models $\mathcal{G}$ directly as the store models by transferring all the model parameters. However, using these models without any fine-tuning as the target models $\mathcal{T}$, could potentially result in reduced forecast accuracy as only sales patterns that are similar across all stores in each state are learned. Therefore, we fine-tune these target models $\mathcal{T}$ to capture the unique sales patterns per store. While the original RSLLMs use 5.3 years of training days per time series, we fine-tune the target models per store using only one year of store-specific training data. This still provides training data for every day of the year. We fine-tune using the `refit function`, which is the only available prebuilt LightGBM function that is capable of adjusting pretrained LightGBM models. The `refit function` keeps the nodes of the decision trees fixed, but recalculates the prediction of every leaf with the latest store-specific training data. The `refit function` does not fully replace the pretrained predictions, but calculates a weighted average between the pretrained and the newly computed prediction values. These weights are determined using one additional hyperparameter, i.e., the decay rate.[11] A decay rate close to 1.0 places more weight on the pretrained prediction values of the leaves of the source models, while a decay rate close to 0.0 focuses mostly on the newly computed prediction values, which are computed using the latest store-specific training data. The STRUT fine-tuning strategy of Segev et al. (2016) suggests to only use the store-specific training data for the computation of the prediction values, if enough training data is available. This translates to a decay rate equal to 0.0, which means that only the newly computed prediction values of the leaves are used, and the pretrained prediction values of the source models are overwritten.

To validate this suggestion empirically, we select the optimal value for the decay rate using a

[9]https://github.com/Mcompetitions/M5-methods
[10]https://www.kaggle.com/c/m5-forecasting-accuracy
[11]https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.Booster.html

time-based cross-validation (CV) strategy that is similar to the one of the winning method.[12] This CV strategy uses the last four months of the M5 training data as four out-of-sample validation sets of 28 days each, which we label CV1 to CV4 (see Figure 1). The decay rate that returns the best forecast accuracy on average for CV1 to CV4, is used to predict the sales of the M5 test set. Note that we optimize the decay rate independently of the other parameters as we simply reuse the model architecture, input features and hyperparameters of the RSLLMs. The results of this CV strategy show that the best performing decay rate is indeed equal to 0.0.

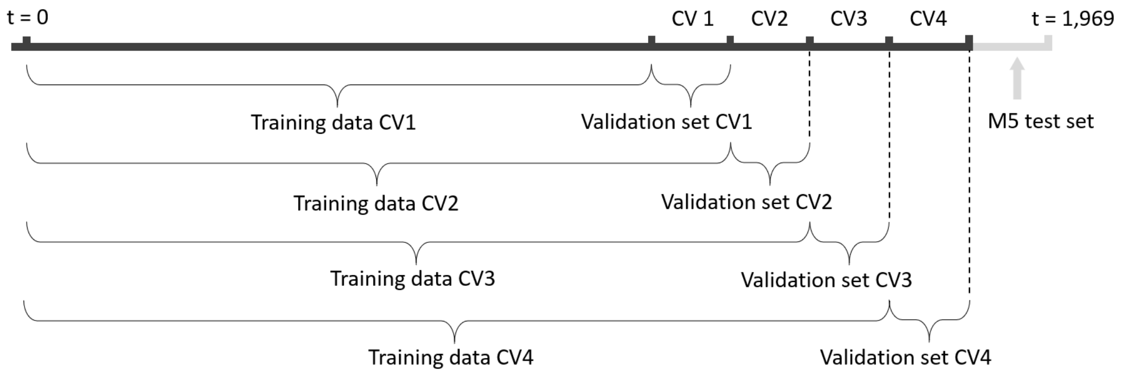

Figure 1: Visualization of the cross-validation (CV) strategy to determine the decay rate. The time-based CV strategy divides the M5 training data in four separate training sets and four different 28-days out-of-sample validation sets, which we label CV1 to CV4. The decay rate that gives the best forecast accuracy on average for CV1 to CV4, is used to predict the sales of the M5 28-days test set.

5.2. Results

Figure 2 visualizes the training time and the forecast accuracy of the RSLLMs, as well as the results of the TL framework with and without fine-tuning for the M5 test set. The training times are measured on a local server with two 24 Core CPUs and 256 GB RAM. The forecast accuracy is measured by the weighted root mean squared scaled error (WRMSSE), the M5 official forecast accuracy metric. As the training of GBDT models are faced with stochasticity, we train the models using five different random seed values.

Figure 2 shows how the application of the TL framework achieves similar levels of forecast accuracy compared to the RSLLMs, while reducing training times by 23.16% to 26.35%. The forecast accuracy of the source models (i.e., TL framework without fine-tuning per store) is on average 4.98% lower than the RSLLMs and the TL framework with fine-tuning. Fine-tuning the source models to identify store-specific sales patterns is thus important to obtain comparable forecast accuracy to the RSLLMs. Without fine-tuning the TL framework training times can be reduced by roughly 70%. If a small reduction in forecast accuracy is acceptable (on the order of 5% in this example), substantial larger reductions in training times can be achieved.

Figure 3 visualizes the WRMSSE on the test set for each of the 12 hierarchical levels, using different random seed values. We find that the TL framework outperforms the RSLLMs at the more aggregate levels (i.e., level 1, level 2, ...), but slightly underperforms at the more disaggregate levels (i.e., level 9, level 10, ...). On average, the TL framework never outperforms the RSLLMs by more than 4.78%, and never underperforms more than 0.95%. Without fine-tuning, however, the source models consistently underperform at most up to 10% at each hierarchical level.

[12]https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/163684

The results above are also consistent for CV1, CV2, CV3 and CV4 with a decay rate of 0.0. As we used these out-of-sample test sets to validate the value of the decay rate, we decided to not report the results in the paper to have a more ‘correct’ evaluation.

6. Conclusions, limitations, discussion and the future

The M5 Accuracy competition was an important milestone for the use of ML in forecasting. It was the first M-competition where pure ML methods outperformed simple statistical methods. Despite their relatively straightforward implementation, they do require *substantial* computation power. We demonstrate how transfer learning (TL) can be applied in hierarchical forecasting to reduce the training time of LightGBM forecasting models, without negatively impacting the forecast accuracy. Our TL framework requires little tuning or coding: It makes use of the hierarchy in the time series, a prebuilt fine-tuning function and the exact same code files of the M5 top-performing methods. Our results indicate that for the M5 limited dataset, a reduction of roughly 25% in training time can be achieved, without affecting the forecast accuracy. We conjecture that even higher reductions in training time can be achieved for settings with thousands of products and dozens of stores in numerous states. Without the fine-tuning step, training times can be reduced by 70%, yet, at the expense of a small reduction in forecast accuracy.

Our TL framework keeps the computational cost of the prediction phase constant, and does not increase the total number of model architectures, input features and hyperparameters as the decay rate can be set to 0.0. Therefore, the computational cost of the experimental phase remains the same, and can even be reduced as our TL framework reduces the speed of retraining models with different model variations. The latter holds only true if retailers optimize the model architecture, input features, and hyperparameters directly with the TL framework. Moreover, the TL framework can be extended to other ML methods such as neural networks (e.g., Mukherjee et al., 2018), and is not limited to hierarchical time series. As an example, the winning method of the Corporación Favorita Grocery Sales Forecasting[13] competition combined multiple neural networks and LightGBM models to forecast the sales of thousands of products in 54 Ecuadorian grocery stores for the next 16 days. Different models were used to forecast each day of the forecast horizon. In this case, instead of grouping time series based on the hierarchy, our TL framework can train source models $\mathcal{G}$ that include multiple days of the forecast horizon, e.g., forecasting four days instead of one, to reduce the total number of models that need to be fully trained.

A reduced training time not only facilitates the practical application to put these models into production, it also comes with financial savings. Assuming that each RSLLM covers roughly 3,000 products per store/region, and each model (with additional optimizations) can be trained in 10 minutes on a 32 CPU machine with 32GB RAM (available on Google Cloud at a rate of \$0.791488 per hour).[14] A large retailer with 1,000 stores (as a comparison, Walmart has 10,500 stores,[15] and Tesco currently operates 4,673 stores)[16] each selling 100,000 products (Nikolopoulos and Petropoulos, 2018) that retrains its models every two weeks (26 times per year), requires to train $26 \times 1,000 \times 100,000/3,000 = 866,666$ models per year. If this retailer would implement an ensemble of similar sized LightGBM models, such as the M5 winning method, it would need to

[13]https://www.kaggle.com/c/favorita-grocery-sales-forecasting
[14]https://cloud.google.com/compute/all-pricing E2 high-CPU machine types us-central1 (accessed 15 July 2021)
[15]https://corporate.walmart.com/our-story/our-locations (accessed 15 July 2021)
[16]https://www.statista.com/statistics/238667/tesco-plc-number-of-outlets-worldwide/ (accessed 15 July 2021)

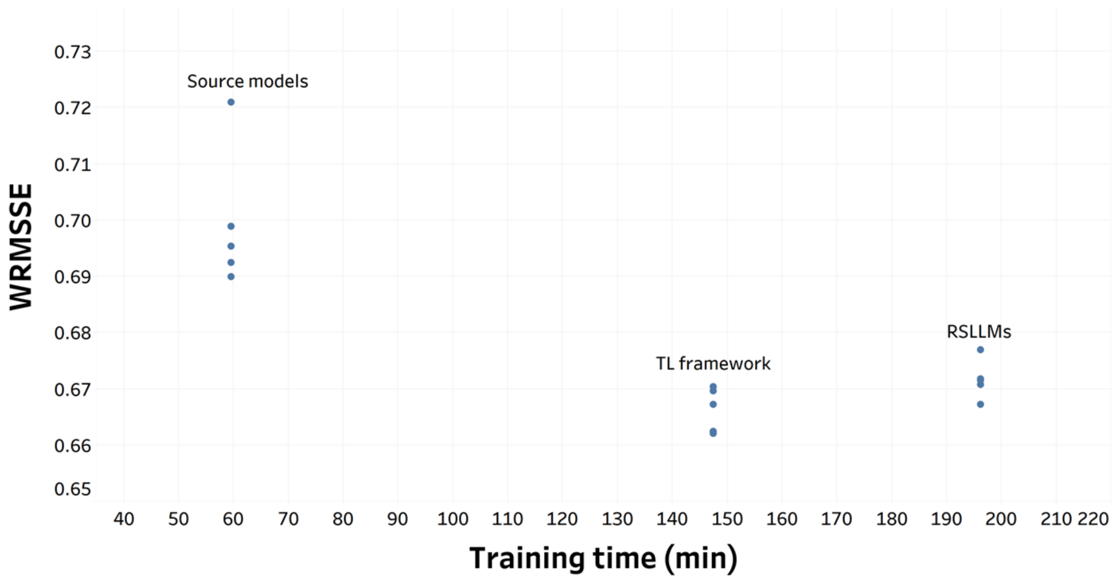

Figure 2: Results of the numerical illustration for the M5 test set. The transfer learning (TL) framework replicates similar levels of forecast accuracy (as measured by the weighted root mean squared scaled error (WRMSSE)) to the recursive store level LightGBM models (RSLLMs) with roughly 25% less training time. The TL framework without fine-tuning (source models) achieves roughly 70% lower training times, at the expense of a 4.98% lower forecast accuracy. Each model is calculated five times with varying random seed values.

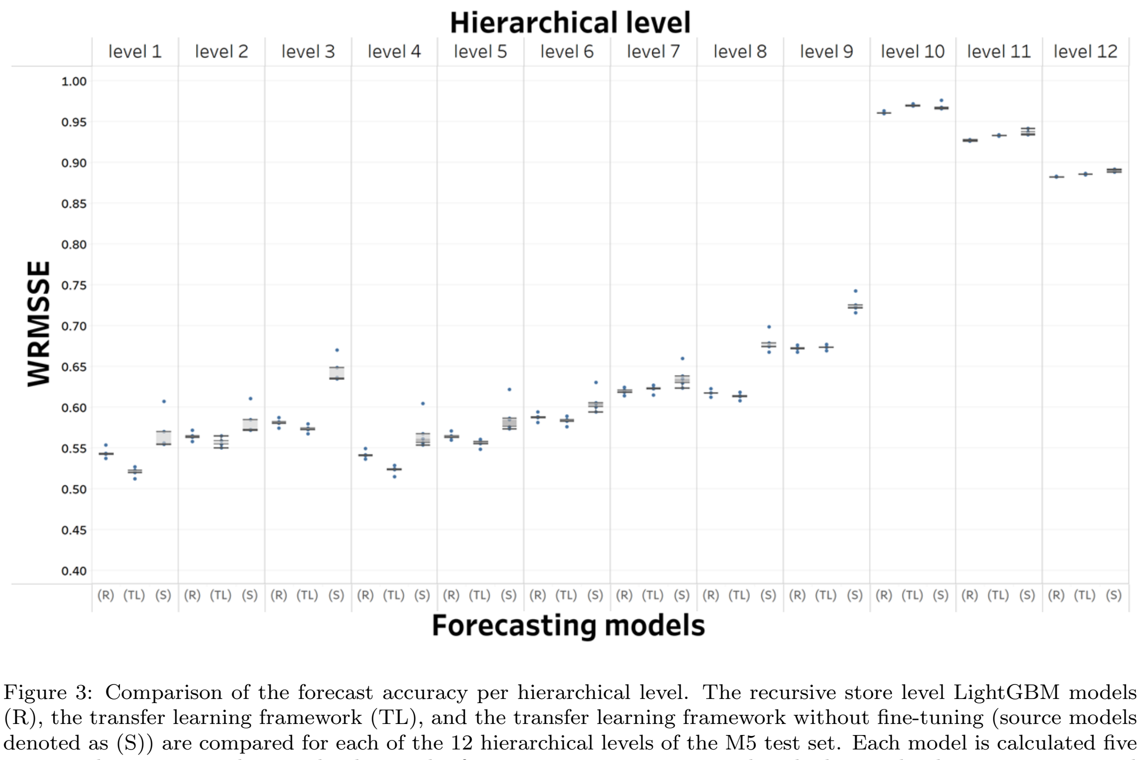

Figure 3: Comparison of the forecast accuracy per hierarchical level. The recursive store level LightGBM models (R), the transfer learning framework (TL), and the transfer learning framework without fine-tuning (source models denoted as (S)) are compared for each of the 12 hierarchical levels of the M5 test set. Each model is calculated five times with varying random seed values. The forecast accuracy is measured with the weighted root mean squared scaled error (WRMSSE). The (TL) outperforms the (R) at the aggregated levels, but slightly underperforms at the more disaggregated hierarchical levels. The source models (S) systematically underperform.

train 5,199,996 models per year,[17] i.e., 866,666 training hours at a total annual cost of \$685,956. A reduction of 25% training time would result in \$171,489 annual savings. In the case of an online retailer, the savings could be even more substantial as these may offer over 100 million products in a wide variety of regions (Seaman, 2018).

Despite the positive results reported in the paper, we acknowledge some limitations and potential improvements of the TL framework that require further research. One limitation is that there is no theoretical guarantee that our implementation of the framework reaches acceptable levels of forecast accuracy. While the `refit function` succeeds in fine-tuning the target models $\mathcal{T}$ in our numerical illustration, its ability to adjust existing model parameters is limited as the `refit function` cannot recalculate any nodes of a decision tree. This may hinder the target models $\mathcal{G}$ to learn any specific patterns at grouping level g, if the time series at g' are too dissimilar from the ones at grouping level g. We believe that this inability also causes the under- and outperformance when comparing the forecast accuracy per hierarchical level. One solution could be to only group time series by g', if they show enough similarity, using advanced clustering-based ensemble models (Godahewa et al., 2020). Another possibility is to test different fine-tuning strategies that are capable of fine-tuning a subset of the nodes. These strategies already exist today (Segev et al., 2016; Fang et al., 2019; Son et al., 2015), but are not yet prebuilt into the LightGBM package. Other improvements may be realized by replacing the random sampling procedure to reduce the total amount of training data of the source models with a more sophisticated sampling approach (e.g., the row and column selection algorithm of Ali and Yaman, 2013). Such an approach could help selecting a smaller and more informative set of training instances to further improve the forecast accuracy and to decrease the computational requirements.

Another limitation is the lack of software optimizations of the `refit function`. Additional experiments show that the `refit function` is relatively inefficient regarding training times. Inefficient training times, together with the lack of other software optimizations (such as not supporting .bin files for faster loading times[18] or using three times more RAM memory during our numerical illustration), suggest that the `refit function` is not as optimized as training LightGBM models from scratch. Therefore, optimizing the `refit function` appears as a promising area of future research, given its potential to further reduce the training time of the TL framework and even improve our results further. Note that we used the latest version of the LightGBM package (version 2.3.1.) that was available during the M5 Accuracy competition.

With our paper we hope to draw the attention of researchers and practitioners to the importance of computational efficiency of forecasting methods, especially if they are intended to be used for hundreds of stores and thousands of products. We believe that this would help the forecast community and commercial software packages to fully embrace the state-of-the-art and leverage the potential of these models for implementation in practice.

References

Afrin, K., Nepal, B., and Monplaisir, L. (2018). A data-driven framework to new product demand prediction: Integrating product differentiation and transfer learning approach. *Expert Systems*

[17]It is worth noting that the winning method trained 220 models for 10 stores with 3,049 products. This resulted in six models per time series. For simplicity, we assume that the computational requirements of 60 RSLLMs (thus six models per store) require a similar amount of training time as training these 220 models, which include global models per store-category and store-department.

[18]https://buildmedia.readthedocs.org/media/pdf/lightgbm/latest/lightgbm.pdf

with Applications, 108:246–257.

Ali, Ö. G. and Yaman, K. (2013). Selecting rows and columns for training support vector regression models with large retail datasets. *European Journal of Operational Research*, 226(3):471–480.

Ashouri, M., Shmueli, G., and Sin, C.-Y. (2019). Tree-based methods for clustering time series using domain-relevant attributes. *Journal of Business Analytics*, 2(1):1–23.

Assimakopoulos, V. and Nikolopoulos, K. (2000). The theta model: A decomposition approach to forecasting. *International Journal of Forecasting*, 16(4):521–530.

Bandara, K., Hewamalage, H., Liu, Y.-H., Kang, Y., and Bergmeir, C. (2020). Improving the accuracy of global forecasting models using time series data augmentation. *arXiv preprint arXiv:2008.02663*.

Bojer, C. S. and Meldgaard, J. P. (2021). Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting*, 37(2):587–603.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Crone, S. F., Hibon, M., and Nikolopoulos, K. (2011). Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. *International Journal of Forecasting*, 27(3):635–660.

Fang, W., Chen, C., Song, B., Wang, L., Zhou, J., and Zhu, K. Q. (2019). Adapted tree boosting for transfer learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 741–750. IEEE.

Fildes, R., Ma, S., and Kolassa, S. (2019). Retail forecasting: Research and practice. *International Journal of Forecasting*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, pages 1189–1232.

Gibney, E. (2017). Self-taught AI is best yet at strategy game Go. *Nature*, 10(1):68–74.

Godahewa, R., Bandara, K., Webb, G. I., Smyl, S., and Bergmeir, C. (2020). Ensembles of localised models for time series forecasting. *arXiv preprint arXiv:2012.15059*.

Guo, C. and Berkhahn, F. (2016). Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*.

Hirt, R., Kühl, N., Peker, Y., and Satzger, G. (2020). How to learn from others: Transfer machine learning with additive regression models to improve sales forecasting. *arXiv preprint arXiv:2005.10698*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hyndman, R. J. (2020). A brief history of forecasting competitions. *International Journal of Forecasting*, 36(1):7–14.

Januschowski, T., Gasthaus, J., Wang, Y., Salinas, D., Flunkert, V., Bohlke-Schneider, M., and Callot, L. (2020). Criteria for classifying forecasting methods. *International Journal of Forecasting*, 36(1):167–177.

Karb, T., Kühl, N., Hirt, R., and Glivici-Cotruta, V. (2020). A network-based transfer learning approach to improve sales forecasting of new products. *arXiv preprint arXiv:2005.06978*.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154.

Makridakis, S., Andersen, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., Newton, J., Parzen, E., and Winkler, R. (1982). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, 1(2):111–153.

Makridakis, S., Chatfield, C., Hibon, M., Lawrence, M., Mills, T., Ord, K., and Simmons, L. F. (1993). The M2-competition: A real-time judgmentally based forecasting study. *International Journal of Forecasting*, 9(1):5–22.

Makridakis, S. and Hibon, M. (1979). Accuracy of forecasting: An empirical investigation. *Journal of the Royal Statistical Society: Series A (General)*, 142(2):97–125.

Makridakis, S. and Hibon, M. (2000). The M3-competition: Results, conclusions and implications. *International Journal of Forecasting*, 16(4):451–476.

Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2018a). The M4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802–808.

Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2018b). Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3):e0194889.

Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2020a). The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74.

Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2020b). The M5 Accuracy competition: Results, findings and conclusions. *International Journal of Forecasting*.

Mukherjee, S., Shankar, D., Ghosh, A., Tathawadekar, N., Kompalli, P., Sarawagi, S., and Chaudhury, K. (2018). ARMDN: Associative and recurrent mixture density networks for eRetail demand forecasting. *arXiv preprint arXiv:1803.03800*.

Nikolopoulos, K. and Petropoulos, F. (2018). Forecasting for big data: Does suboptimality matter? *Computers & Operations Research*, 98:322–329.

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

Petropoulos, F. and Grushka-Cockayne, Y. (2021). Fast and frugal time series forecasting. *Available at SSRN 3792565*.

Seaman, B. (2018). Considerations of a retail forecasting practitioner. *International Journal of Forecasting*, 34(4):822–829.

Segev, N., Harel, M., Mannor, S., Crammer, K., and El-Yaniv, R. (2016). Learn on source, refine on target: A model transfer learning framework with random forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9):1811–1824.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1):75–85.

Son, J., Jung, I., Park, K., and Han, B. (2015). Tracking-by-segmentation with online gradient boosting decision tree. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3056–3064.

Spiliotis, E., Makridakis, S., Semenoglou, A.-A., and Assimakopoulos, V. (2020). Comparison of statistical and machine learning methods for daily SKU demand forecasting. *Operational Research - An International Journal*, pages 1–25.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7).

Torrey, L. and Shavlik, J. (2010). Transfer learning. In *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques*, pages 242–264. IGI global.

Waldrop, M. M. (2016). The chips are down for Moore's law. *Nature News*, 530(7589):144.