Ben-Gurion University of the Negev

Faculty of Engineering Sciences

Department of Industrial Engineering and Management

Stochastic Weight Averaging in long-term time-series forecasting

Thesis submitted in partial fulfillment of the requirements

for the M.Sc. degree in the Faculty of Engineering Sciences

By: **Yuval Siman-Tov Levy**

April 17, 2024

Ben-Gurion University of the Negev

Faculty of Engineering Sciences

Department of Industrial Engineering and Management

Stochastic Weight Averaging in long-term time-series forecasting

Thesis submitted in partial fulfillment of the requirements

for the M.Sc. degree in the Faculty of Engineering Sciences

By: **Yuval Siman-Tov Levy**

Under the supervision of **Dr. Yakir Brenchko and Dr. Omri Azencot**

Author: [illegible] Date: 17/04/2024

Supervisors: [illegible] [illegible] Date: 17/04/2024

Chair of Graduate Studies Committee:

[illegible] Date: 17/04/2024

April 17, 2024

Ben-Gurion University of the Negev

Master of Sciences Thesis

Yuval Siman-Tov Levy

April 2024

Stochastic Weight Averaging in long-term time-series forecasting

Abstract

The recent boom of forecasters is fueling the ongoing pursuit for improvements through yet more architectural modifications. Ensemble learning stands as a popular alternative, renowned for its ability to oftentimes outperform any of its base-models. Indeed, deep ensembles have been applied successfully in time-series forecasting through various methods. Albeit, despite their considerable gains, prevailing methods suffer major computational overhead preventing them from widespread use. Stochastic Weight Averaging (SWA) is an ensemble method that averages weights along the optimization trajectory. Since its release, SWA has had many follow-up papers in a wide range of applications, to the best of our knowledge, excluding forecasting. This work, evaluates the performance of SWA on three selected transformer-based architectures for long-term time-series forecasting for the first time.

Keywords

Ensemble learning, Stochastic Weight Averaging, Time-series, Long-term forecasting, Optimization trajectory, Loss-landscape.

Acknowledgements

I am heartily thankful to my advisors, Dr. Yakir Brenchko and Dr. Omri Azencot, for their excellent instruction, clear guidance and mentoring, and unlimited supply of wisdom whenever needed. It has been a real honor to work with such two great researchers from different areas of data science.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Making decisions based on the input of multiple people or experts has been a common practice in human civilization. Over the past few decades, researchers in the machine learning community have studied schemes that share such a joint decision procedure. These schemes are generally referred to as ensemble learning.

Ensemble learning aggregates several individual models to obtain a combined model with better generalization, outperforming every single model in it. These model can either be of the same architecture (homogeneous ensemble) or of different types (heterogeneous ensemble). A necessary and sufficient condition for an ensemble of learners to be more accurate than any of its individual members is if these learners are accurate (an accurate learner is one that has an error rate better than random guessing on new x values) and diverse (two learners are diverse if they make different errors on new data points), but these are two conflicting goals [1]. Increasing the accuracy of learning machines will inevitably sacrifice the diversity of learning machines, and increasing the diversity of learning machines will reduce the accuracy of learning machines. How to produce an optimal combination strategy is the core content of ensemble learning research.

Ensemble methods' functionality was theoretically supported by theories like bias–variance [2, 3], strength correlation [4], stochastic discrimination [5], margin theory [6] and the bias–variance–covariance decomposition [7]. Seminal algorithms such as Random Forests (RF) [4] and XGBoost [8], staples in machine learning, are among the most popular choices for tabular data by practitioners [9, 10].

Indeed, ensemble learning has been effective in many tasks including medical diagnosis [11], fraud detection [12], sentiment analysis [13] and recommendation systems [14] - with usage even in Unsupervised [15] and Reinforcement Learning [16] settings. Yet some domains still remain relatively under-researched.

Time series (TS) analysis is a highly important research field, encompassing various domains of science, engineering, and finance. Common tasks are anomaly detection, data imputation and most notably forecasting (short-term/long-term/probabilistic). Time series forecasting (TSF) is an essential part of real-world applications, such as weather forecasting, energy consumption, and financial assessment. In all, one pressing demand is to extend the forecast time into the far future, which is quite meaningful for pre-planning and early warning, furthermore modern incarnations often exhibit large panels of related time series, all of which need to be forecasted simultaneously.

Although still widely used, traditional time series forecasting models, such as State Space Models (SSMs) and Autoregressive (AR) models, are designed to fit each time series independently. Besides, they also require practitioners' expertise in manually selecting trend, seasonality and other components [17]. These issues make deep learning especially appealing for usage. Among deep learning models, Transformer-based models have achieved great success on various application fields such as natural language processing (NLP), computer vision (CV), and speech, benefiting from the attention mechanism which can automatically learn the connections between elements in a sequence, thus becoming ideal for sequential modeling tasks.

However, deep learning is not a panacea, as models typically have high variances and low biases [18]. Moreover, in a complicated environment, it is difficult for an individual deep learning model to maintain high forecasting accuracy and robustness. Ensemble learning, on the other hand, has proved to be an effective method for overcoming this problem [19]. In this work we build upon 3 contemporaneous Transformer architectures - PatchTST, FEDformer, Autoformer [20–22] - and test whether ensemble learning can improve their results in the long-term forecasting task.

The introduction of deep learning into the ensemble framework combines the advantages of both paradigms and offers a simple boost to a single network's performance [23]. Ensemble methods have been well studied in recent years and applied widely in different applications becoming ubiquitous, both in practice and the academic literature, because of their conceptual simplicity and ease of implementation. For example, while XGBoost [24] has long been used as a state-of-the-art method for tabular data, the NODE architecture [25] showed even better results. In addition, as of 2022 the top performing models on ImageNet [26] use ensembles to achieve their highest score. Deep ensembles also provide good uncertainty quantification and are widely used as a benchmark to beat, achieving the best performance on out-of-distribution uncertainty [27].

Many methods for constructing ensembles have been developed throughout the years. In *Bagging* [28], several subsets of the data are generated by uniformly sampling with replacement from the only available dataset, then, individual models are trained on each separate subset, with their outputs averaged in the end. A similar method, albeit slightly more complex, is *Boosting* [29], for which several variants exist, here only Adaboost is described: In boosting, models are built in sequence. In each iteration the current bootstrap sampling distribution is dependent on the previous iteration where boosting adjusts the bootstrap sampling distribution by weighting misclassified points more heavily. At the end, boosting weights each model's vote dependent on their misclassification rate.

Another traditional form of ensemble method is motivated by the Bayesian perspective, in which the available data is believed to be the only data (hence averaging over different datasets is not applicable). In Bayesian inference the best individual model is the model that maximizes the posterior probability (aka MAP estimate). This approach, though, ignores the remaining uncertainty of the posterior distribution. To incorporate all uncertainty into the model selection process, under *Bayesian Model Averaging* (BMA) [30] a single model is constructed by averaging over the predictions of all possible models weighted by their posterior probability. It is essentially a weighted ensemble over the predictions of all models in the hypothesis space.

When the data space can be divided into multiple homogeneous regions, a *Mixture of Experts* is applicable, wherein each learner becomes an expert on a sub-space by employing a special error function, then, a gating network is to compute the probability of assigning each example to each expert. At the same time as the experts are learning to handle their assigned examples, the gating network is learning which expert should be assigned each example based on the relative discriminative performance of the experts for that example. This is a form of *Stacked Generalization* in which base learners are combined hierarchically through a meta-leraner estimating the weights of the outputs from each base predictor.

Unfortunately, although a deep ensemble provides large accuracy boosts relative to individual models, it requires storing M models (consisting of at least millions of parameters each) and preforming M forward passes at prediction time - making predictions using a whole ensemble of models cumbersome and too computationally expensive, especially in environments in which computational constraints are limited, where even a linear scaling of storage requirements can be prohibitive.

In order to overcome these constraints, *Knowledge Distillation* (KD) [31] aims at compressing all M models in an ensemble into a single model, "distilling their knowledge". The key concept behind knowledge distillation is the teacher–student network framework, where the teacher networks are selected from a pool of pre-trained networks and the student network distils knowledge of multiple teachers into a single and often simpler network trained to match the distribution of the much larger teachers. The testing phase is storage and computationally efficient, as the samples only need to pass through a single student network.

Negative Correlation attempts to train and combine the individual networks in the same learning process. In negative correlation, all the individual networks are trained simultaneously and interactively through the correlation penalty terms in their error functions. Rather than producing unbiased individual networks whose errors are uncorrelated, negative correlation can create negatively correlated networks to encourage specialisation and cooperation among the individual networks.

Another line of works attempting to reduce the training cost is called "*implicit* ensemble learning", in which a single model is trained in such a manner that it "mimics" an ensemble of multiple networks, without incurring additional cost. Thus, the training time of an ensemble is the same as the training time of a single model. A notable work in this field is *Dropout* [32], creating an ensemble network by randomly dropping out hidden nodes from the network during the training of the network. In dropout, the models share parameters, with each model inheriting a different subset of parameters from the parent neural network. This parameter sharing makes it possible to represent an exponential number of models with a tractable amount of memory. Typically a tiny fraction of the possible sub-networks are each trained for a single step, and the parameter sharing causes the remaining sub-networks to arrive at good settings of the parameters. Furthermore, the training set encountered by each sub-network is a subset of the original training set sampled with replacement, much like the bagging algorithm. Generalization of DropOut was given in Wan et al. [33] with *DropConnect*. Unlike DropOut which drops each output unit, DropConnect randomly drops each connection and hence, introduces sparsity in the weight parameters of the model. Similar to DropOut, DropConnect creates an implicit ensemble during test time by dropping out the connections (setting weights to zero) during training. In *Stochastic depth* [34], for each mini-batch during training, a subset of layers are randomly dropped and bypassed with the identity function.

At a high level, Dropout is both at once a regularizer and a training procedure, motivated to approximate an ensemble. In this work we explore a different method aiming at implementing these high level ideas: **Stochastic Weight Averaging** (SWA) [35].

Chapter 2

Related Work

2.1 Deep Ensemble Learning for Time-Series Forecasting

2.1.1 Stacked Generalization

Stacking is an umbrella term referring to the process of training one model on top of the predictions of the base learners (in the simple mean and median cases, the stacking model is constant rather than learned). Theoretical guarantees for stacked generalizations were given in [36], with applications for time-series forecasting recently seen in [37].

For example, EnsemLSTM [38] traines a cluster of Long Short-Term Memory neural networks (LSTM), with diverse hidden layers and neurons to explore and exploit information of wind speed time series. Then predictions of LSTMs are aggregated into a top-layer composed of Support Vector Regression Machine (SVRM) with the Extremal Optimization (EO) algorithm introduced to optimize its parameters. Similarly, Qiu et al. [39] propose to train several Deep Belief Networks (DBN) using different number of epochs and an SVR with inputs as the outputs of the DBNs. Its output is the final prediction.

GEFTS [40] is also homogeneous ensemble, however is meta learner is of the same type of the base learners - a Generalized Regression Neural Network (GRNN). In [41], attentive transfer learning based Graph Neural Networks (GNN) utilize learned prior knowledge to improve the learning process in a spatial-temporal time series prediction task. The attention network is designed to assign weights to outputs from N GNN models learned from source domains together with the output from a base GNN learned from the target domain. The complete model's output is the weighted summation of all N+1 models. EA-DRL [42] also offers an interesting approach which utilizes the deep reinforcement learning framework. A very diverse pool of base models is constructed with the combination strategy modeled as a sequential decision making process. An actor-critic model then aims at learning the optimal policy to aggregate the weights in a continuous action space.

2.1.2 Mixture of Experts

A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem. In [43], each base learner - Gaussian Process Regression (GPR) - finds the optimal balance between accuracy and diversity in a small region of the feature space. Hence, the feature space is segmented into smaller sub-spaces, and each subspace is used as the training set of a GPR. Then, each is trained to provide a balance between the accuracy on its corresponding segment (or training set), and diversity on the adjacent segments.

Wavelet transform (WT) is a commonly used decomposition algorithm. It decomposes the original TS into certain orthonormal sub series by looking at the time frequency domain. Jiajun et al. [44] use WT to disaggregate wind speed series. Then to enhance the forecasting precision of the subseries, DBN was applied to extract the high dimensional features. Besides, to overcome the limitation of the conventional DBN, the forecasts for each subseries, processed by DBN, were executed by the light gradient boosting machine (LGBM).

But the wavelet basis function and decomposition scale need to be determined in advance, which is not an adaptive decomposition method. This is not the case with the Empirical Mode Decomposition (EMD) method which does not need to set any basis functions. EMD is another popular method of decomposition based on the signal characteristics of the data itself. Essentially, the intrinsic fluctuation pattern is obtained by the characteristic time scale of the data. Complex signals can be decomposed into finite Intrinsic Mode Functions (IMFs) and residual representing the trend of change by the EMD. For instance, Qiu et al. [45] decompose their load demand data into IMFs with a DBN learning the rules of each component to build multiple sub-models. Both the original data set and output of all sub-models are combined to correct the prediction error of sub-models. Finally, the prediction results of all IMFs are combined by either a weighted summation to obtain an aggregated output. On the other hand, Fan et al. [46] propose taking the accuracy and diversity of parameters for multi-objective function solved by the Tchebycheff method.

2.1.3 Knowledge Distillation

Seasonal and Trend decomposition using Loess (STL) is another general and robust decomposition algorithm that decomposes a time series into three sub-components: trend, seasonal term, and residual. The sum of the three components approximates the original time series. Lin et al. [47] predict each sub-component based on KD, where LSTM and Gated Recurrent Units (GRU) learn and provide feedback to each other. Finally, predictions are aggregated in 2-stages, first for each sub-component and then for the entire series. Similarly, Floratos et al. [48] aim to simultaneously train the student model with teachers' ensemble. First, they train each of the teacher model using the hard labels l, then, the output of the ensemble is aggregated to form the soft targets that will be used for training of the student model - for one epoch only. these steps are repeated for a total of N training epochs. Campos et al. [49] further extend KD to allow assigning appropriate weights to different teachers, such that their capabilities contribute purposefully to the training of the student model.

2.1.4 Other approaches

Sloughter et al. [50] extend the BMA technique to wind speed data through modeling the component Probability Density Functions (PDF) as untransformed gamma distributions. Member specific parameter estimation was done by linear regression and the remaining by maximum likelihood using a variant of the EM algorithm, ECME. In Du et al. [51], the Bayesian Optimization (BO) algorithm was applied in hyperparameter tuning phase to automatically search for the best combination configuration and combine the top-performing model candidates, from a diverse pool of base models, on a time-varying basis. OEP-ROC [52] doesn't focus on how to combine individual models in an ensemble, but how to select them, i.e. *Ensemble Pruning*. They employ gradient-based saliency maps for online ensemble pruning of DNNs. pTSE [53] is an ensemble method for probabilistic forecasting based on Hidden Markov Model (HMM). Assuming that at each time t, there exists an optimal model in the ensemble, they treat the collection of member models as a hidden state space, where the distribution of each observation is determined by its hidden state. assume that at each time t, there exists an optimal model in the ensemble. The whole ensemble step is achieved by inferring the stationary distribution of the HMM.The empirical distribution is estimated via incorporating Mixture Quantile Estimation (MQE) with the Baum-Welch algorithm.

2.2 Weight Averaging

Studies have shown the existence of multiple local minima in training neural networks, where some enjoy better generalizability than others [54]. This has inspired researchers to take advantage of the diversity of multiple local minima for new ensemble techniques. In [55] for example, authors utilize the non-convex nature of the loss landscape and the ability of stochastic gradient descent (SGD) to converge and escape from local minima, to develop SNapshot Ensemble (SNE).

In SNE a single model is trained by a cyclic learning rate scheduler, where the learning rate is abruptly changed every few epochs to perturb the network, and thus may lead to diversity exploiting both good and bad local minima found on the optimizing path. Converging M-times when the model reaches a minimum, multiple versions of the model are saved during the training process. These snapshots are then averaged to form an ensemble. Instead of training multiple deep learners independently from scratch, the training time here is the same as that of the single model. We note however that the need of cyclic learning rates in SNE makes it incompatible to Transformer [56] which requires a warm-up and inverse square root learning rate.

Garipov et al. [57] observed that the local optimum of modern deep neural networks are connected by paths (represented by by very simple curves) with lower loss values in the loss plane, thus, different networks can be found through relatively small steps in the weight space. Fast Geometric Ensembling (FGE) samples multiple nearby points in weight space to create high performing ensembles. FGE uses a high frequency cyclical learning rate with

SGD to generate a sequence of points that are spatially close to each other in the weight space, but produce diverse predictions for the ensemble. As a result, multiple base learners can be obtained at training time equivalent to that of a single model, which reduces the time overhead comapred to traditional ensemble methods.

Although SNE and FGE can effectively reduce the time of training, in order to make full use of them, the obtained learners need to be stored. While they provide higher accuracy, their inference is slow as the predictions from many models have to be averaged at test time to form the final ensemble.

Observing that the weights of the networks ensembled by FGE are on the periphery of low-value area on the loss plane, never reaching its central points, Izmailov et al. [35] propose Stochastic Weight Averaging (SWA), leading to a wider optima by averaging the weights proposed over SGD iterations. Since the difference between averaging predictions and averaging weights is of the second order of smallness, by averaging weights instead of predictions, SWA reduces the time and space overhead for the testing stage.

Although the general idea of maintaining a running average of weights traversed by SGD was first considered in convex optimization dating back to Ruppert [58], Polyak and Juditsky [59], this procedure is not typically used to train neural networks. Acting both as a regularizer and a training procedure, SWA shows simple averaging of multiple points along the trajectory of SGD, leads to even better generalization performance - so much so, it has been natively supported in PyTorch [60].

Stephan et al. [61] show that under several simplifying assumptions, running

SGD with a constant learning rate is equivalent to sampling from a Gaussian distribution centered at the minimum of the loss, and the covariance of this Gaussian is controlled by the learning rate. Following this explanation, points proposed by SGD can be interpreted as being constrained to the surface of a sphere, since they come from a high dimensional Gaussian distribution. Hence SWA is effectively able to go inside the sphere in order to find higher density solutions.

SWA-Gaussian (SWAG) [62] is an extension for Bayesian uncertainty and calibration by estimating a distribution over model weights. Providing an average, SWA solution serves as the first moment of a fitted Gaussian. Together with a covariance matrix also derived from the SGD iterates, BMA is performed via sampling from this Gaussian distribution. Similarly, Izmailov et al. [63] use first principal components from the SGD trajectory together with SWA solution as a shift vector, to construct low-dimensional subspaces and (also) preform BMA.

A wide basin found by SWA is particularly relevant for BMA, since flat regions of the loss represent a large volume in a high-dimensional space, dominating the overall average. However, SWAG focus their approximation on only a single basin, making yet a limited contribution to computing the entire Bayesian predictive distribution. By representing multiple basins of attraction, a better approximation posterior can be derived. MultiSWAG [64] combines multiple independently trained SWAG approximations to create a mixture of Gaussians approximation to the posterior, where each Gaussian is centerd on a different basin of attraction.

SWA in Low-Precision (SWALP) [65], as the name suggests, is an implementation in scenarios were fewer bits are used to represent numbers, allowing scalability, memory savings, portability, and energy efficiency. A wide optimum found by SWA is also especially relevant in the context of low-precision training as it is more likely to contain high-accuracy low-precision solutions. In SWALP, quantization is made for the weights, activations, backpropagation errors, and gradients.

SWA in Parallel (SWAP) [66] is a modification to distributed settings. Instead of sampling multiple models from a sequence generated by SGD, multiple nodes compute estimates simultaneously on disjoint subsets of the mini-batch. At first large-batches are used to compute an approximate solution quickly, results are refined by each worker using smaller-batches. The transition point between the large-batch and small-batch is an added hyper-parameter. Workers produce a consensus estimate by averaging all estimates, with one synchronization event per iteration.

In semi-supervised setting, the optimization trajectories of the popular Π [67] and Mean Teacher [68] consistency-based models was studied, finding that SGD struggles to converge on the consistency loss and continues to make large steps that lead to changes in predictions on the test data. fast-SWA [69], is proposed to train them in order to accelerate convergence by averaging multiple points within each cycle of a cyclical learning rate schedule.

SWA has also been applied by Nikishin et al. [70] to model-free environments in RL allowing to reduce the effect of noise on training and improve stability.

Chapter 3

Method

3.1 Problem Definition

Matrices, vectors and scalars are denoted by uppercase bold, lowercase bold and lowercase normal letters, i.e., $\mathbf{X}$, $\mathbf{x}$ and x, respectively.

A time series consists of sequential data with values associated with time intervals. For time series containing C variates, given historical data $\mathbf{X} = \{\mathbf{x}_1^{(t)}, \ldots, \mathbf{x}_C^{(t)}\}_{t=1}^{L}$, wherein L is the *lookback* window size and $\mathbf{x}_k^{(t)}$ is the value of the k^{th} variate at time step t, the multivariate long-term time series forecasting task is to predict the values $\hat{\mathbf{X}} = \{\hat{\mathbf{x}}_1^{(t)}, \ldots, \hat{\mathbf{x}}_C^{(t)}\}_{t=L+1}^{L+T}$, at the $T \gg 1$ future (discrete) time steps, called the *horizon*. Multivariate data is transformed into univariate data by treating each feature of the sequence as an individual time series, i.e. C=1. Future values are of a single target $\{\hat{\mathbf{x}}^{(t)}\}_{t=L+1}^{L+T}$.

Let $f(\mathbf{X}; \mathbf{w}_i)$ denote the predictions of a neural network, parametrized by weights vector $\mathbf{w}_i$[1] of epoch $i \in [1, n]$. The SWA solution is given by by averaging weights of several epochs in the weight space: $\hat{\mathbf{X}} \equiv f(\mathbf{X}; \overline{\mathbf{w}})$.

3.2 Base Models

Here, representing f, are three transformer-based architectures. Autoformer [22] adopts a segmentation-based representation mechanism. It devises a simple seasonal-trend decomposition architecture with an auto-correlation mechanism working as an attention module. The auto-correlation block measures the time-delay similarity between inputs signal and aggregates the top-m similar sub-series to produce the output with reduced complexity. Frequency Enhanced Decomposed transformer (FEDformer) [21] applies attention operation in the frequency domain with Fourier transform and wavelet transform. It achieves a linear complexity by randomly selecting a fixed-size subset of frequency. Here we use the Fourier based model. PatchTST [20] utilizes channel-independent where each channel contains a single univariate time series that shares the same embedding within all the series, and a patching design which segments the channel into sub-series level patches that serve as input tokens to Transformer. There are also two versions of PatchTST. PatchTST/64 implies the number of input patches is 64, which unlike the default look-back window of 96 in Autoformer and FEDformer, uses a look-back window of 512. PatchTST/42 means the number of input patches is 42, which has the lookback window of 336. For memory considerations we only evaluate here PatchTST/42.

[1]weight matrices are layer-wise flattened and appended, so a single vector is formed.

3.3 Stochastic Weight Averaging

Apart from minor adjustments, our work takes the off-the-shelf Pytorch implementation. SWA can be trained with either a cyclical or a high constant learning rate. We follow the latter default scheduler. Training procedure starts the same as the original training, with $\mathbf{w}_i$ as the model's weights. After SWA starts, at epoch s, the original optimization trajectory changes according to the SWA scheduler. Hereafter the model's weights are denoted as $\mathbf{p}_i$ (for *parameters*). In addition, swa-model is a deep copy initialized with $\mathbf{p}_s$. Its weights/parameters vector is the running average μ_i:

$$\forall i \geq s \quad \mu_i \equiv \frac{\mathbf{p}_i + (i-s)\mu_{i-1}}{1+(i-s)} \tag{3.1}$$

Weights are typically equally averaged, though custom strategies may also be applied. Originally, the training process is early stopped within 3 times to avoid overfit (*patience*), however to conduct valid comparison, we don't limit here the number of training epochs, as SWA also trains until the last epoch. At test time, swa-model is used for inference.

In PatchTST, batch normalization is used, and so, an additional pass is run after the training is finished, to compute the running mean and standard deviation of the activations for each layer. For other architectures, this is redundant as layer norm is used instead. The number of epochs in which the learning rate is increasing to its high constant phase is called the anneal epochs (for PatchTST, we adjust it to anneal per batch, as per its unique scheduler). This increase can be of either a linear shape or cosine shape.

Chapter 4

Experiments

4.1 Configurations and Parameter Settings

All models are trained with L2 loss, using ADAM optimizer with an initial learning rate of 10^{-4}. The experiments are repeated on three different random seeds, implemented using PyTorch 2.0 with Python 3.10 and conducted on a single NVIDIA A6000 48GB GPU. Prediction length varies in 96, 192, 336, 720. Configurations that differ between models are listed in 4.1, with the rest, uniquely define per model. For the full list see [20–22].

For SWA, the learning rate is ought to be high enough to enforce exploration, so it was set to 10^{-4}, as per the initial learning rate of all models. We start SWA mid-training, at epoch 6, to balance between the pre-averging and averaging periods of the process. We also set the annealing epochs to 3 in order to mitigate any rapid changes caused by the SWA scheduler (for convenience, these are 60 and 30, respectively, in case of PatchTST). We deem

Config.	Autoformer/FEDformer	PatchTST
Encoder	2	3
Model Dim	512	128
FFN Dim	2048	256
Heads Dim	8	16
Dropout	0.05	0.2
Train Epoch	10	100
Batch Size	32	128

Table 4.1: Configuration differences in base models

these three hyperparametes the most important, hence we test their impact on results further in the appendix. Annealing and averaging strategies are of minor importance according to some prior checks we've conducted, thus will remain fixed to 'cosine' and 'equal averaging', respectively, throughout this work.

4.2 Data Sets

Our study is conducted on two extensively utilized real-world datasets from distinct domains: Electricity Transform Temperature[1] (ETT) and Weather[2] 4.2. The datasets are picked from those that the tested models are measured on, and are available for public use. ETT (hereafter, ETTm2) dataset contains data collected from an electricity transformer every 15 minutes between July 2016 and July 2018, and includes seven features of load and one series of oil temperatures (target variable in the univariate setting). Weather has been recorded every 10 minutes for the whole year of 2020 in Weather Sta-

[1]https://github.com/zhouhaoyi/ETDataset
[2]https://www.bgc-jena.mpg.de/wetter/

tion of the Max Planck Biogeochemistry Institute, Germany. It contains 21 meteorological indicators, such as air temperature, humidity, etc. We follow standard protocol and split all datasets into training, validation and test set in chronological order by the ratio of 6:2:2 for the ETTm2 dataset and 7:1:2 for the weather dataset, to ensure no data leakage issues.

Dataset	Timesteps	Variates	Frequency
ETTm2	69,680	7	15 min
Weather	52,696	21	10 min

Table 4.2: Dataset details

4.3 Performance Measures

Following conventions, we use Mean Squared Error (MSE) and Mean Absolute Error (MAE), calculated across all time steps and variables, as the core metrics to compare performance (recall, C=1, in the univariate case):

$$MSE \equiv \frac{1}{T \cdot C} \sum_{t=L+1}^{L+T} \sum_{k=1}^{C} (\mathbf{x}_k^{(t)} - \hat{\mathbf{x}}_k^{(t)})^2, \quad MAE \equiv \frac{1}{T \cdot C} \sum_{t=L+1}^{L+T} \sum_{k=1}^{C} |\mathbf{x}_k^{(t)} - \hat{\mathbf{x}}_k^{(t)}| \tag{4.1}$$

Thus, the overarching goal is to test whether -

$$MSE(f(\mathbf{X}; \mu_n), \mathbf{X}) < MSE(f(\mathbf{X}; \mathbf{w}_n), \mathbf{X}) \tag{4.2}$$

and/or

$$MAE(f(\mathbf{X}; \mu_n), \mathbf{X}) < MAE(f(\mathbf{X}; \mathbf{w}_n), \mathbf{X}) \tag{4.3}$$

.

4.4 Results

We summarize the experimental results on multivariate and univariate forecasting tasks in Tables 4.3, 4.4, respectively. For each experiment we report the mean and Standard Deviation (SD) of MSE and MAE over 3 runs.

Multivariate results Though relatively minor, our results indicate slightly inferior forecasting capabilities of the FEDformer and PatchTST models trained with SWA, yielding an average increase of 3.25% and 1% in MSE, respectively (in MAE, 2% and 0.5% on average, respectively). Specifically, FEDformer under SWA, seems to struggle more with horizon 96, attaining 7.2% and 5.7% MSE deterioration rates on weather and ETTm2, respectively (4.34% and 3.33% respectively, in MAE). In contrast, when ensembling Autoformer weights, SWA maintains a leading position across most datasets and horizons, reducing MSE and MAE by 1.5% and 0.6%, on average, respectively. Notably there is a performance difference in horizon 720 between the two datasets. The model's MSE worsened by 7% when trained on the weather dataset, but improved by the same percentage when trained on ETTm2. These represent its worst and best scores among all horizons. Overall, SWA seems comparable to conventional training procedure for the multivariate forecasting task.

Univariate results As for the univariate setting, the performance of SWA is evidently subpar to all regular model versions. When used in PatchTST, it marks an average deterioration 78.43% in MSE, while both in Autoformer and in FEDformer it exhibits a colossal 109% free-fall in average performance. While not as inflated, MAE depicts a similar picture. For Autoformer, as in the multivariate task, we spot divergent outcomes in a certain horizon, now it's 96: while training Autoformer on the weather dataset led to a 436.36% increase in MSE, its performance improved by 30.5% when trained on ETTm2, making it the strongest improvement for this task across models, datasets, and horizons. We find SWA to hinder FEDformer and PatchTST most in the weather dataset (for ETTm2, the slide isn't as steep). Most notably, for horizon 192, the latter soars from a mean MSE score of 0.0256 to 0.1161. This while for horizon 96, FEDformer achieves nothing short of a Brobdingnagian result with a **706.36**% ($0.011 \rightarrow 0.0887$) in MSE deterioration.

We evaluate SWA's performance based not only on the experiments' means, but also on their variance, considering whether SWA improvements (deteriorations) still hold even after the SD is added to (subtracted from) the mean. For SWA to demonstrate superiority over standard training we look for this "statistical significance" across either architectures, datasets, horizons or forecasting tasks. Alas, no such pattern emerge. 'Autoformer/ETTm2/96' was the only experiment exhibiting such improvement. In contrast, 'FEDformer/Weather/96' was the only experiment showing consistent deterioration in both performance measures, for both forecasting type.

In conclusion, in the absence of a visible improvement pattern, and given the negligible performance upgrades compared to the humongous deteriorations, we deduce this ensemble scheme failed to catch up with ordinary training for the tested architectures in the evaluated datasets.

Model	Dataset	Horizon	Original		SWA	
			MSE	MAE	MSE	MAE
Autoformer	ETTm2	96	0.2723 ± 0.0125	0.3381 ± 0.0069	**0.258 ± 0.0069**	**0.3311 ± 0.0038**
		192	0.3265 ± 0.0122	0.365 ± 0.007	**0.3165** ± 0.0152	**0.3621** ± 0.0112
		336	0.3882 ± 0.022	**0.3985** ± 0.0076	**0.3847** ± 0.0196	0.3994 ± 0.0092
		720	0.5625 ± 0.0752	0.4919 ± 0.0413	**0.5238** ± 0.0322	**0.4793** ± 0.023
	weather	96	0.3237 ± 0.0328	0.3785 ± 0.0238	**0.3146** ± 0.0331	**0.3738** ± 0.0215
		192	**0.3197** ± 0.0119	**0.376** ± 0.0096	0.3223 ± 0.0117	0.3781 ± 0.0094
		336	0.3756 ± 0.0067	0.4043 ± 0.005	**0.3706** ± 0.0039	**0.4018** ± 0.0046
		720	**0.5955** ± 0.2777	**0.5143** ± 0.1357	0.6372 ± 0.3237	0.5242 ± 0.1478
FEDformer	ETTm2	96	**0.207** ± 0.0081	**0.2972** ± 0.0066	0.2188 ± 0.0116	0.3071 ± 0.0104
		192	**0.2563** ± 0.0001	**0.3226** ± 0.0013	0.2569 ± 0.0019	0.3233 ± 0.002
		336	0.3365 ± 0.0087	0.3749 ± 0.0033	**0.3334** ± 0.0061	**0.3736** ± 0.002
		720	**0.4421** ± 0.0157	**0.4377** ± 0.0064	0.4434 ± 0.0205	0.4391 ± 0.0095
	weather	96	**0.2347 ± 0.015**	**0.3109 ± 0.0128**	0.2516 ± 0.0111	0.3244 ± 0.0099
		192	**0.2833 ± 0.0079**	**0.3421 ± 0.0062**	0.298 ± 0.0124	0.3548 ± 0.0113
		336	**0.3424** ± 0.0146	**0.3864** ± 0.0076	0.3641 ± 0.0293	0.4029 ± 0.0168
		720	**0.4714** ± 0.0536	**0.4681** ± 0.0381	0.4806 ± 0.0546	0.4724 ± 0.0381
PatchTST	ETTm2	96	**0.1876** ± 0.0016	**0.2709** ± 0.0016	0.1897 ± 0.0026	0.2729 ± 0.0017
		192	0.2707 ± 0.005	**0.3282** ± 0.0031	**0.2703** ± 0.0056	0.3282 ± 0.0032
		336	**0.3367 ± 0.0038**	**0.3694 ± 0.0034**	**0.3428** ± 0.0025	0.3734 ± 0.0015
		720	**0.4276 ± 0.0016**	**0.4277 ± 0.0026**	0.4364 ± 0.0048	0.4326 ± 0.0038
	weather	96	**0.1519** ± 0.004	**0.1996** ± 0.0003	0.1524 ± 0.001	0.1998 ± 0.0012
		192	**0.2076** ± 0.004	**0.2502** ± 0.0004	0.2089 ± 0.0021	0.2509 ± 0.0011
		336	**0.2725 ± 0.0015**	**0.2946 ± 0.0002**	0.2768 ± 0.0029	0.2965 ± 0.0015
		720	**0.3441** ± 0.0035	**0.3464** ± 0.0018	0.3459 ± 0.0061	0.3472 ± 0.0051

Table 4.3: Multivariate forecasting results.

The mean of the best results for each experiment are **bolded**.
SD is also bolded if the result is "statistically significant".

Model	Dataset	Horizon	Original		SWA	
			MSE	MAE	MSE	MAE
Autoformer	ETTm2	96	0.1669 ± 0.0533	0.2916 ± 0.0201	**0.1177 ± 0.0174**	**0.2592 ± 0.0202**
		192	0.1683 ± 0.0208	0.3124 ± 0.0147	**0.1554 ± 0.0121**	**0.3017 ± 0.0095**
		336	0.1958 ± 0.0393	0.3398 ± 0.0342	**0.187** ± 0.0397	**0.3328** ± 0.0345
		720	**0.2533** ± 0.0459	**0.3727** ± 0.0233	0.311 ± 0.1507	0.4052 ± 0.0781
	weather	96	**0.011** ± 0.0067	**0.0776** ± 0.0171	0.059 ± 0.0783	0.1602 ± 0.1088
		192	**0.0051 ± 0.001**	**0.0542 ± 0.0054**	0.0152 ± 0.0054	0.0895 ± 0.0226
		336	**0.0052 ± 0.0005**	**0.053 ± 0.0025**	0.0074 ± 0.0012	0.0629 ± 0.0057
		720	**0.0054** ± 0.001	**0.0555** ± 0.0028	0.0172 ± 0.0187	0.0734 ± 0.022
FEDformer	ETTm2	96	**0.0661 ± 0.0005**	**0.1928 ± 0.0008**	0.0679 ± 0.0012	0.1959 ± 0.0017
		192	**0.0982** ± 0.0019	**0.2391** ± 0.0034	0.1002 ± 0.0034	0.242 ± 0.0033
		336	**0.1374** ± 0.0125	**0.285** ± 0.0121	0.1444 ± 0.0216	0.2926 ± 0.0227
		720	**0.1879** ± 0.0036	**0.3361** ± 0.0043	0.1901 ± 0.0089	0.339 ± 0.011
	weather	96	**0.011 ± 0.007**	**0.0869 ± 0.0332**	0.0887 ± 0.022	0.2651 ± 0.0463
		192	**0.0247** ± 0.0249	**0.1276** ± 0.0795	0.0582 ± 0.056	0.2009 ± 0.1307
		336	**0.0042** ± 0.0021	**0.0506** ± 0.0136	0.005 ± 0.0018	0.0559 ± 0.0104
		720	0.0076 ± 0.0005	0.0704 ± 0.0034	**0.0075** ± 0.0014	**0.0691** ± 0.0064
PatchTST	ETTm2	96	0.0908 ± 0.0019	0.2168 ± 0.0014	**0.0894** ± 0.0045	**0.2156** ± 0.002
		192	**0.1351** ± 0.0012	0.2706 ± 0.001	0.1351 ± 0.0076	**0.2703** ± 0.0043
		336	**0.1783** ± 0.0134	0.3155 ± 0.0123	0.1784 ± 0.004	**0.3145** ± 0.0037
		720	**0.2221** ± 0.0046	**0.3674** ± 0.0035	0.2242 ± 0.0058	0.3687 ± 0.0043
	weather	96	**0.0038 ± 0.0015**	**0.0421 ± 0.0066**	0.0132 ± 0.0064	0.0682 ± 0.0116
		192	**0.0256 ± 0.0062**	**0.0709 ± 0.011**	0.1161 ± 0.0669	0.1151 ± 0.0384
		336	0.0435 ± 0.0356	0.0808 ± 0.0211	**0.0353** ± 0.0142	**0.0773** ± 0.0211
		720	**0.0666** ± 0.0352	**0.0692** ± 0.0046	0.0972 ± 0.0523	0.0825 ± 0.0066

Table 4.4: Univariate forecasting results.

The mean of the best results for each experiment are **bolded**.
SD is also bolded if the result is "statistically significant".

4.5 Analysis

Shedding light on the effect of SWA, we analyse results from these 3 experiments, representatives of the different possible outcomes:

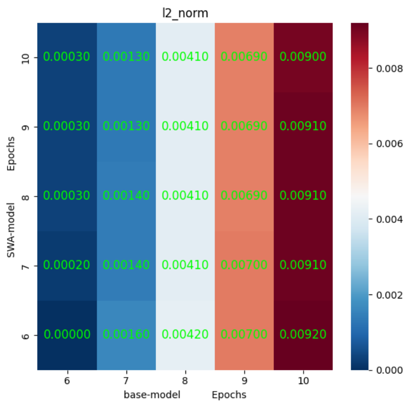
- Autoformer/ETTm2/M/96
  (”statistically significant” improvement over the baseline)
- PatchTST/ETTm2/M/192
  (performance, relatively, on par with baseline)
- FEDformer/weather/S/96
  (”statistically significant” deterioration over the baseline)

These experiments where chosen to cover as many configurations as possible, to enable a more comprehensive comparison with original training (all are included apart from the longer horizons). For each experiment we include an evaluation of different stages in training using a series of 6 heat maps with both L2 norm and cosine similarity. The left hand side evaluates the relative Euclidean distance between each pair of vectors, while the the right figures measure the cosine similarities between them - 4.1, 4.2, 4.3. Of note, this comparison is meaningless before averaging starts, hence only epochs s to n are incorporated.

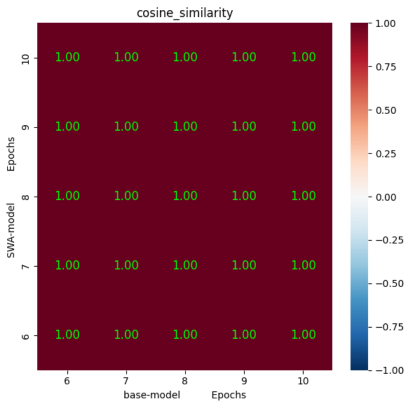
SWA explorers the space in one direction alone - the same direction as the original trajectory, i.e $\forall i, j \quad S_C(\mathbf{w}_i, \mathbf{p}_i) = S_C(\mathbf{p}_i, \mathbf{p}_j) = 1$. This pattern is shared by all experiments indicating it's agnostic of the loss-landscape (see plots 4.1(b), 4.2(b), 4.3(b) and 4.1(d), 4.2(d), 4.3(d)). As expected, in each step weights move further away from their starting value - 4.1(c), 4.2(c), 4.3(c). However, notice the relative distances provided, at least for the 'formers', these distances are still small. Given the aligned directions, plots 4.1(a), 4.2(a), 4.3(a) can be seen as the projection of $\mathbf{p}_i$ onto the original set of weights $\mathbf{w}_i$, which also differs very little. Meaning, for these two models, every $\mathbf{p}_i$ is also approximately the same size as $\mathbf{w}_i$. From prior checks we conducted, this phenomenon is not unique to the horizon, it stems from the architectures themselves, regardless of other hyper-parameter.

Let $\mathbf{g}_i$ denote the gradient vector w.r.t weight $\mathbf{p}_i$. In plots 4.1(f), 4.2(f), 4.3(f) we compare the cosine similarity between each pair. SWA is designed to explore the loss landscape around the solution found at regular training. Despite the near orthogonality of gradients between each different epoch pairs, in Autoformer and PatchTST, it is evident the gradients' magnitude still isn't enough to change the optimization trajectory seen by the cosine similarity between weights. And so, still there is little exploration in space. Even more so, the gradients in FEDformer aren't even different in directions producing an even less diverse ensemble. Plots 4.1(e), 4.2(e), 4.3(e) are describing the relative magnitude of differences between pairs of gradients. Nonetheless, since there is no monotonic trend visible, they convey little additional information about training. Even the seemingly near uniform image of ratios in FEDformer and PatchTST, suggesting a relatively stable optimization trajectory, may mislead. In each, a single epoch is in fact a outlier to the rest distorting the true picture. Consider epoch 76 for example, it is so far away, all other look very close to each other in comparison, so much so, omitting it might tell a different story. Gradients sizes at epoch 10 suggest however that the allocated time wasn't enough for SWA to converge.

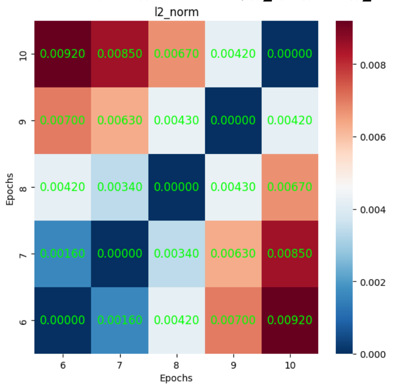
Subsequently, in 4.4 we plot linear slices of the loss landscape connecting the baseline and numerous SWA final solutions, corresponding to different values for s. We plot the vaues of the loss function along the line connecting these to points. $\lambda = 0$ represents weights of the model trained using SWA (μ_n), while $\lambda = 1$ represents weight vector at the end of the original training ($\mathbf{w}_n$). Two models that converge to different minima will likely have a non-convex interpolation, with a spike in error when λ is between 0 and 1, as in 4.4(b). One can also notice the curvature around solutions in 4.4(c), where there is a steep drop in loss in either of the close vicinities. This differs from 4.4(a) wherein the shift is less sharp, meaning SWA solutions reside in a wider basin. For the latter, the effect of s, is the most prominent as the smallest ensemble sizes exhibit a completely different pattern than the rest.



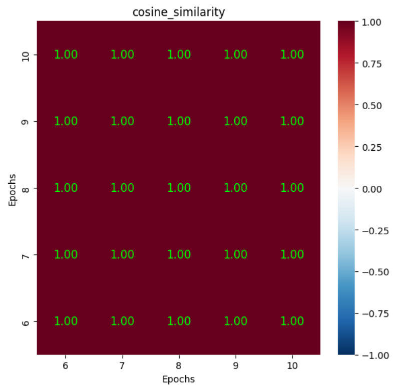
(a) $\|\mathbf{w}_i - \mathbf{p}_i\|_2 \, / \, \|\mathbf{w}_i\|_2$



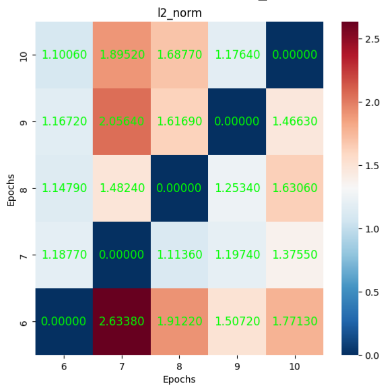
(b) $S_C(\mathbf{w}_i, \mathbf{p}_i)$



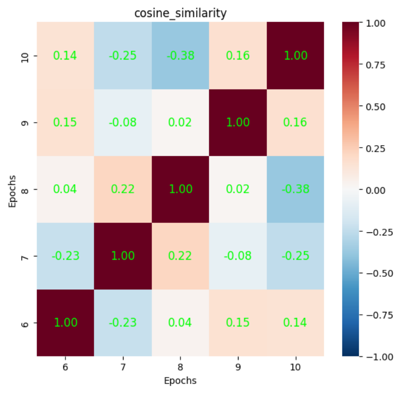
(c) $\left\|\mathbf{p}_i - \mathbf{p}_j\right\|_2 \, / \, \|\mathbf{p}_i\|_2$



(d) $S_C(\mathbf{p}_i, \mathbf{p}_j)$



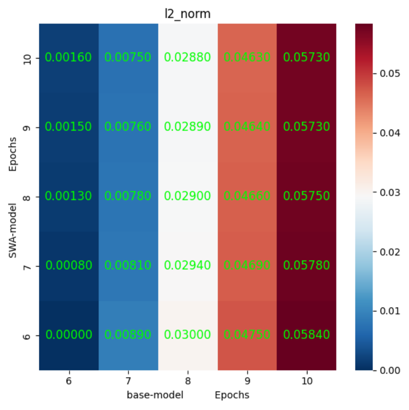
(e) $\left\|\mathbf{g}_i - \mathbf{g}_j\right\|_2 \, / \, \|\mathbf{g}_i\|_2$



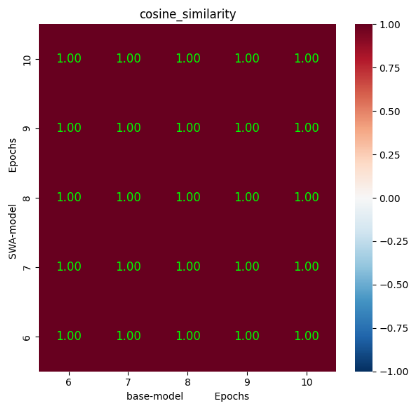
(f) $S_C(\mathbf{g}_i, \mathbf{g}_j)$

Figure 4.1: Autoformer/ETTm2/M/96: A case study of improvement

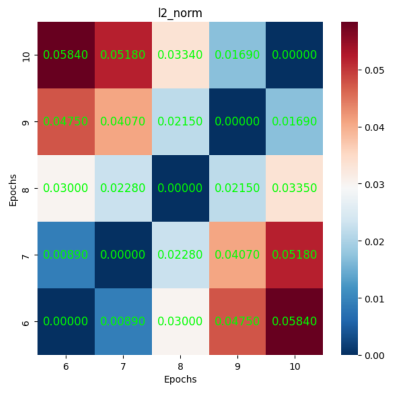
leftt column: Relative distances. **right column**: Cosine similarity. **first row**: original vs. SWA optimization trajectories. **middle row**: weights of the SWA trajectory. **last row**: gradients of the SWA trajectory.



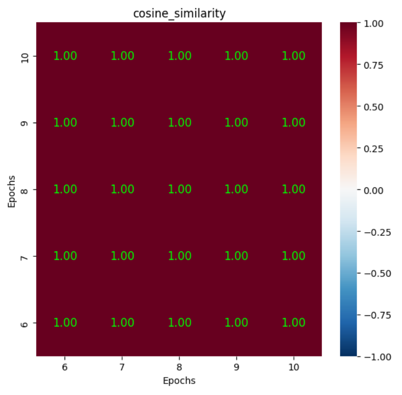
(a) $\|\mathbf{w}_i - \mathbf{p}_i\|_2 \,/\, \|\mathbf{w}_i\|_2$



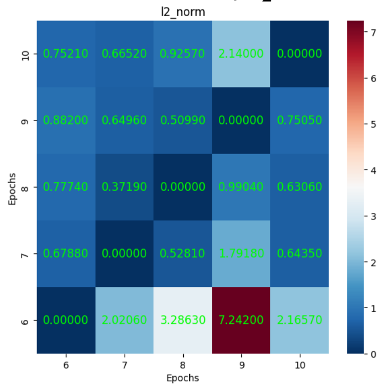
(b) $S_C(\mathbf{w}_i, \mathbf{p}_i)$



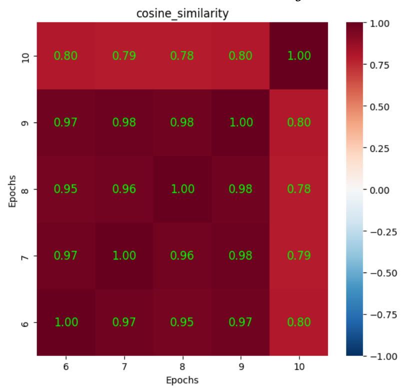
(c) $\left\|\mathbf{p}_i - \mathbf{p}_j\right\|_2 \,/\, \|\mathbf{p}_i\|_2$



(d) $S_C(\mathbf{p}_i, \mathbf{p}_j)$



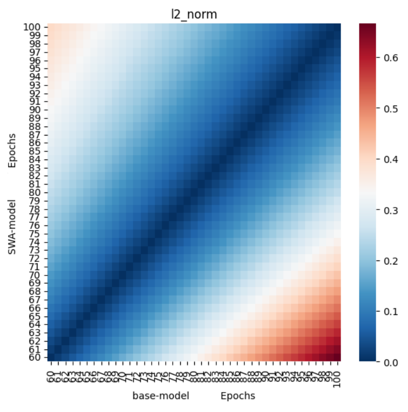
(e) $\left\|\mathbf{g}_i - \mathbf{g}_j\right\|_2 \,/\, \|\mathbf{g}_i\|_2$



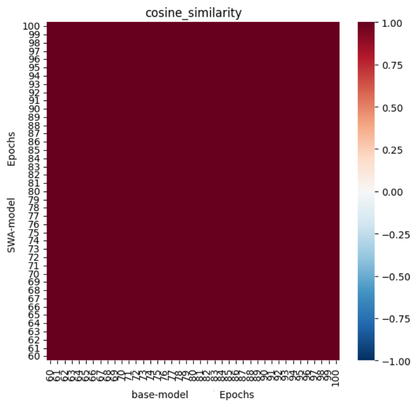
(f) $S_C(\mathbf{g}_i, \mathbf{g}_j)$

Figure 4.2: FEDformer/weather/S/96: A case study of deterioration

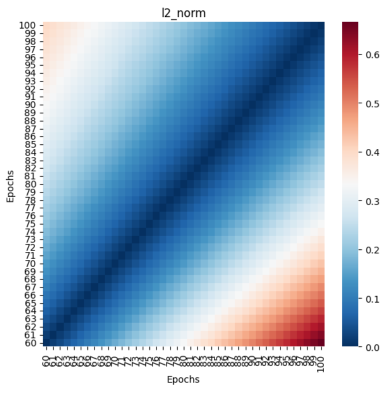
leftt column: Relative distances. **right column**: Cosine similarity. **first row**: original vs. SWA optimization trajectories. **middle row**: weights of the SWA trajectory. **last row**: gradients of the SWA trajectory.



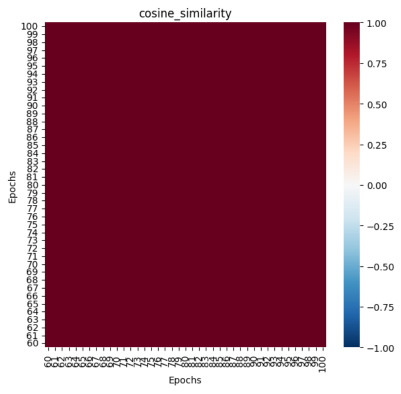
(a) $\|\mathbf{w}_i - \mathbf{p}_i\|_2 \,/\, \|\mathbf{w}_i\|_2$



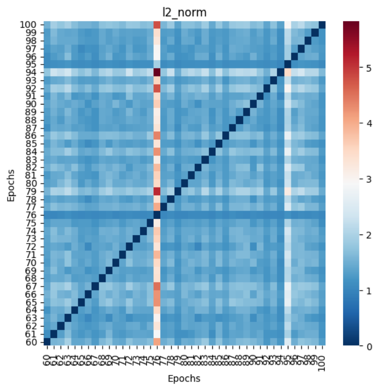
(b) $S_C(\mathbf{w}_i, \mathbf{p}_i)$



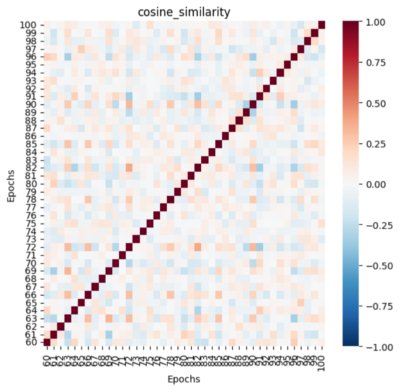
(c) $\|\mathbf{p}_i - \mathbf{p}_j\|_2 \,/\, \|\mathbf{p}_i\|_2$



(d) $S_C(\mathbf{p}_i, \mathbf{p}_j)$



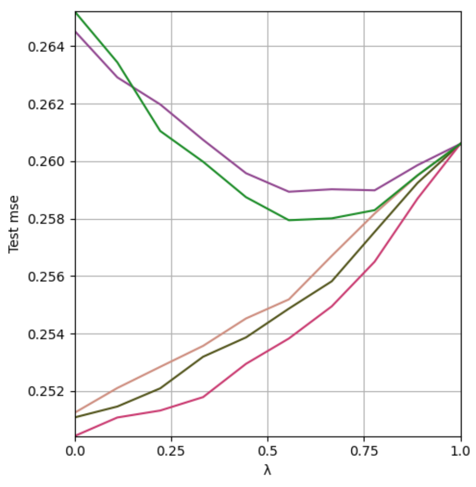
(e) $\|\mathbf{g}_i - \mathbf{g}_j\|_2 \,/\, \|\mathbf{g}_i\|_2$



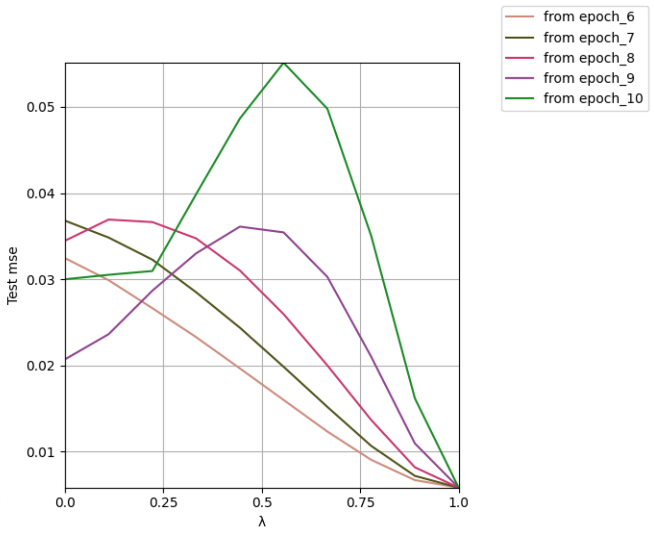
(f) $S_C(\mathbf{g}_i, \mathbf{g}_j)$

Figure 4.3: PatchTST/ETTm2/M/192: A case study of equal performance

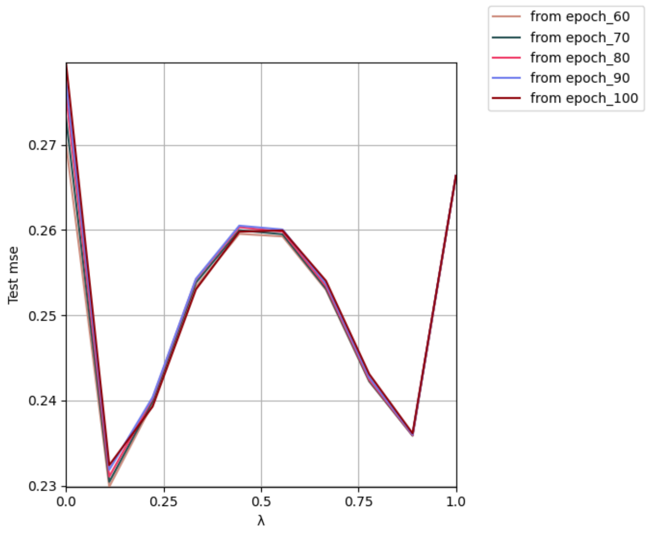
leftt column: Relative distances. **right column**: Cosine similarity. **first row**: original vs. SWA optimization trajectories. **middle row**: weights of the SWA trajectory. **last row**: gradients of the SWA trajectory.



(a) Autoformer



(b) FEDformer



(c) PatchTST

Figure 4.4: All case studies: Interpolation between base and SWA solutions

Test loss along the linear path between the baseline and different SWA solutions, created by different s. The convex combination is given by: $(1-\lambda)\mu_n + \lambda\mathbf{w}_n$.

Chapter 5

Discussion and Conclusions

This thesis extends the current literature on SWA to another domain, yet to be explored: time-series. In long-term forecasting, transformers stand out as powerful apparatus, displaying exceptional capabilities in handling messy datasets from real-world contexts, and so in this work, we have tested SWA on Autoformer, FEDformer and PatchTST architectures, comparing it to the regular training process.

Our results suggest that while the loss has reduced to some extant in several experiments, for the majority, performance was not consistent or significant enough to outperform training in its original form. Even more so, we discovered SWA failed to converge within the allocated epochs conventionally used for training. In fact, for selected set-ups we looked at, this behaviour persisted as far as even 10 times the original time frame. This despite the two optimization trajectories follow exactly the same direction, with the difference in weight being negligible.

An important caveat to these conclusions is that SWA underwent only minor adjustments, and was mostly based on the public implementation offered by Pytorch. In addition, the models at were also naively used in order for the comparison to be as valid as possible. Thus, this is by no means to say SWA is not applicable for forecasting tasks or time-series as a whole.

In any event, these findings merit further study. First and foremost we recommend following our appendix and reevaluate each SWA run with its best hyperparameter configurations, as well as experiment with other datasets and commonly used architectures including linear forecasting models. Secondly, we view favorably a further inspection of the loss-landscape as it severed as a major motivator for the development of SWA in the first place. All weights averaged in SWA stem from the same initialization point and and the same optimization trajectory and are highly correlated. As diversity is key for any ensemble, future work also could offer algorithmic adjustments for a different choice of weights averaged along the path.

Appendix A

Hyperparameter sensitivity

Upon finishing, we dot the i's and cross the t's with a one-way sensitivity analysis for hyperparameters. As stated in 4.1, we explore how the learning rate (lr), anneal epochs (ae) and ensemble size (es, controlled by s) affects the performance of SWA. For a constant learning rate we experiment with lr$\in \{10^{-3}, 10^{-4}, 10^{-5}\}$, and for anneal epochs we try ae$\in \{1, 2, 3\}$. Among ensemble size we trim the first and last 2 epochs and consider all (discrete) s values in between, corresponding to ensemble sizes varying in $[3, 8]$ (for PatchTST everything is multiplied by 10).

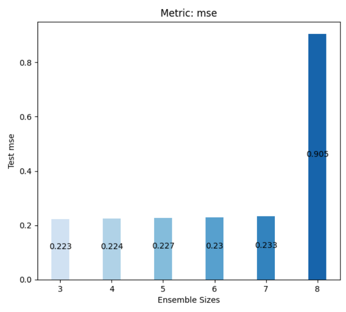
To ease the reader, instead of analysing all possible 1,728 permutations, we follow 4.5 further inspecting the 3 representative case studies. Below are 3 figures, one for each architecture. Each figure is comprised of a 3*3 grid displaying the performance of different ensemble sizes, given each lr-ae combination. In each plot, bars are sorted from the best preforming ensemble size, to the worst (scores for are in black).

To isolate to effect of each hyperparameter, we fix the rest, and measure the variance in loss values - the higher they are, the more sensitive the hypeparameter is to change. To get a sense of what is considered a large variance, we compare each value to the entire distribution of variances from all configurations kept constant. For example, we set lr=10^{-5} and ensemble size=31 to measure the variance of the loss when anneal epochs are varied in PatchTST - this is one value of the distribution.

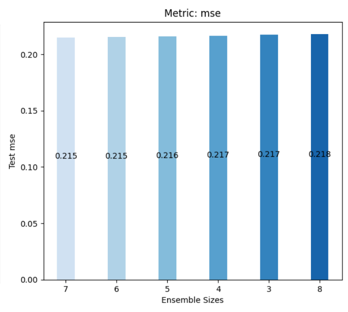
Learning rate It has been reported by Fort and Jastrzebski [71] that setting a too high lr has an adverse effect on SWA, such that is making the weights average lie at a high loss area. We therefore hypothesised the higher the learning rate, the worst the performance is, with high sensitivity to change in values. In practice, only in PatchTST has the best and worst lr repeat themselves under all es-ae combinations, and so we derive conclusions based on the most frequent results. Our conjecture was met only in FEDformer and PatchTST, with Autoformer exhibiting utterly the opposite behaviour. In addition, a sensitive learning rate will be characterized by a large average of variance values, or at least a strong negative skew in them. Distribution of variance values for the 'formers' was the complete opposite with a mean just shy above 0, and a moderate right tail. The distribution for PatchTST was close to normal, with a only a light negative skew. To sum up, for the selected options, the learning rate doesn't seems quite robust, however its sensitivity is dependent in other hyperparameters as well.

Anneal epochs Interestingly, anneal epochs sensitivity is highly correlated with the learning rate - the higher the learning rate, the higher the variance in loss values when ae is changed, regardless of the ensemble sized used. Notwithstanding, ae still doesn't appear to be very sensitive, for any of the models. Here all 3 distributions have the same shape, roughly centered around 0 with a moderately positive skew (medium sized right tail). For all architectures, only 10% of lr-es combination have extreme variance values compared to the rest (further than a single std.). Regarding the best preforming value, in most cases, for FEDformer, 1 (or 10 for Patch) is more suitable. For Autoformer it's generally 2.

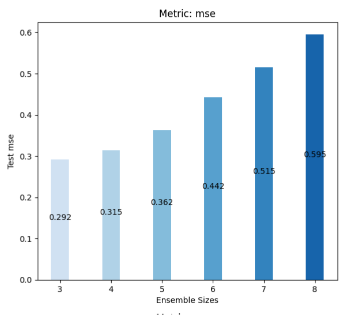
Ensemble sizes Broadly, there is little difference between the 'formers': median of variances is close to 0, and 90% of lr-ae cases with similar variance in ensemble size performance (with some right skew). Also visible is a relatively high correlation between ensemble size sensitivity and the learning rate - for higher rates, there is a greater variance in losses between different ensemble sizes. In Autoformer ensemble size of 3 achieved, for most lr-ae combinations, the lowest MSE - in FEDformer, it was 4. The most frequent worst preforming ensemble sizes where also quite similar - 6 in FEDformer and 7 in Autoformer. On the flip side, in PatchTST, averaging more weights generally gives better performance. In most cases, an ensemble of 71 weights outperforms the rest of options, with 21 as ensemble size, the worst, under all lr-ae permutations. While there is a slight skew to the left in the distribution of variances in losses, the small mean indicates that, similarly to the previous architectures, the starting point for SWA , s, has also minor impact on performance.



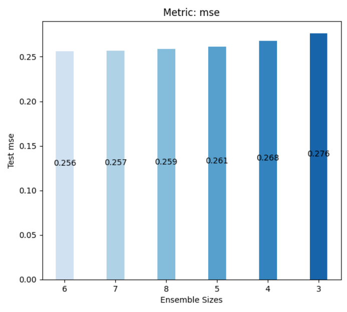
(a) lr $= 10^{-3}$, ae $= 1$



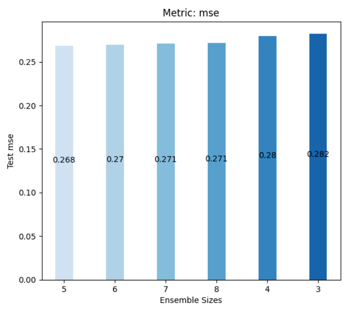
(b) lr $= 10^{-3}$, ae $= 2$



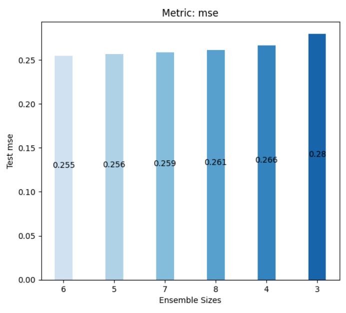
(c) lr $= 10^{-3}$, ae $= 3$



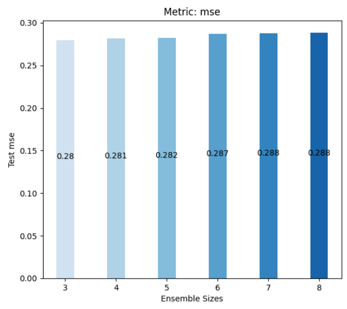
(d) lr $= 10^{-4}$, ae $= 1$



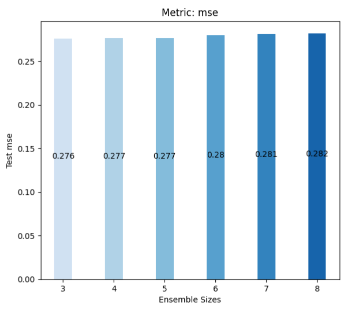
(e) lr $= 10^{-4}$, ae $= 2$



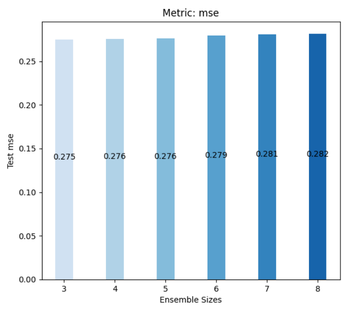
(f) lr $= 10^{-4}$, ae $= 3$



(g) lr $= 10^{-5}$, ae $= 1$



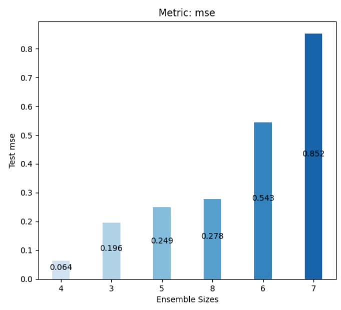
(h) lr $= 10^{-5}$, ae $= 2$



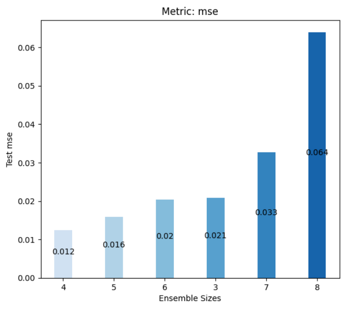
(i) lr $= 10^{-5}$, ae $= 3$

Figure A.1: Autoformer/ETTm2/M/96: Hyperparameters sensitivity

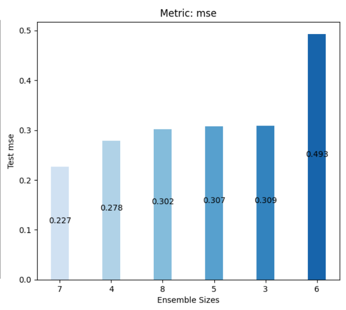
columns: constant learning rate and different anneal epochs.
rows: different learning rates and constant anneal epochs.



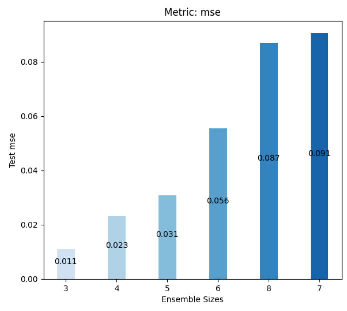
(a) lr $= 10^{-3}$, ae $= 1$



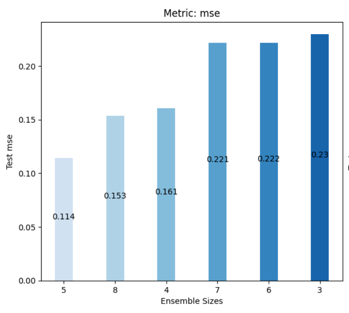
(b) lr $= 10^{-3}$, ae $= 2$



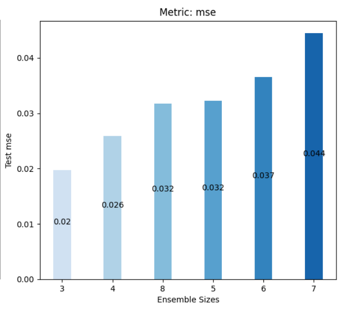
(c) lr $= 10^{-3}$, ae $= 3$



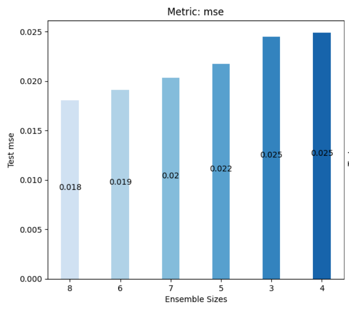
(d) lr $= 10^{-4}$, ae $= 1$



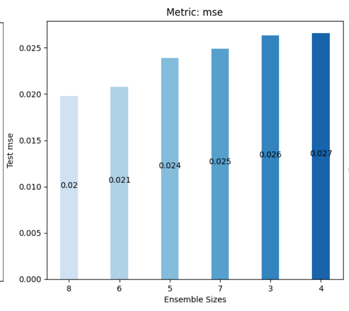
(e) lr $= 10^{-4}$, ae $= 2$



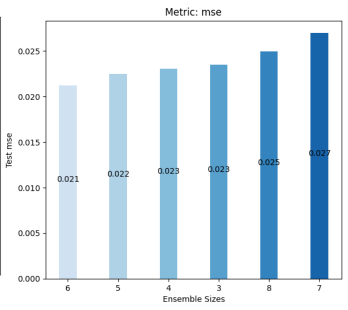
(f) lr $= 10^{-4}$, ae $= 3$



(g) lr $= 10^{-5}$, ae $= 1$



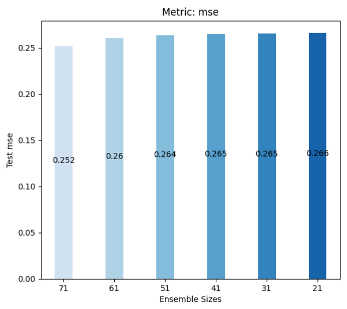
(h) lr $= 10^{-5}$, ae $= 2$



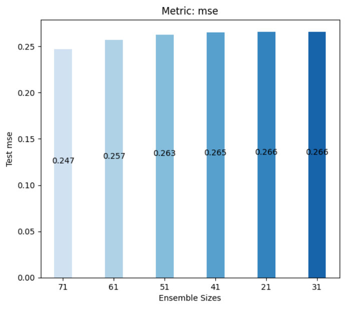
(i) lr $= 10^{-5}$, ae $= 3$

Figure A.2: FEDformer/weather/S/96: Hyperparameters sensitivity

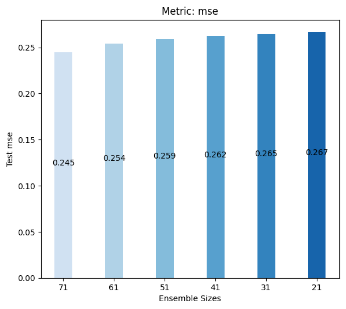
columns: constant learning rate and different anneal epochs.
rows: different learning rates and constant anneal epochs.



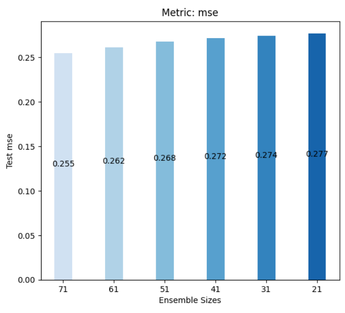
(a) lr $= 10^{-3}$, ae $= 10$



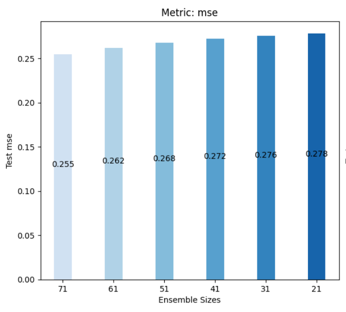
(b) lr $= 10^{-3}$, ae $= 20$



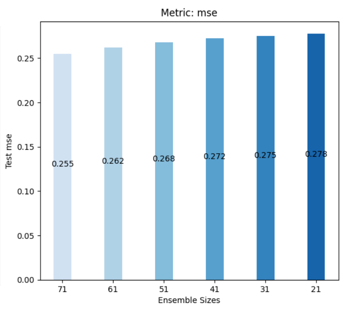
(c) lr $= 10^{-3}$, ae $= 30$



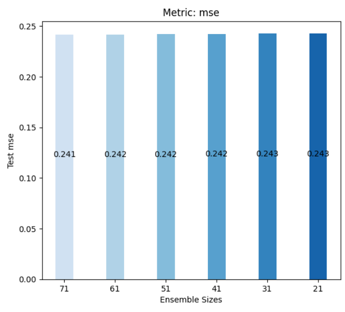
(d) lr $= 10^{-4}$, ae $= 10$



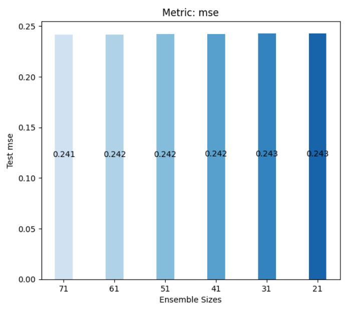
(e) lr $= 10^{-4}$, ae $= 20$



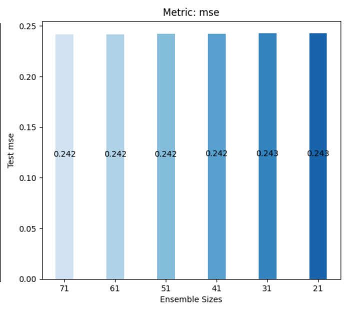
(f) lr $= 10^{-4}$, ae $= 30$



(g) lr $= 10^{-5}$, ae $= 10$



(h) lr $= 10^{-5}$, ae $= 20$



(i) lr $= 10^{-5}$, ae $= 30$

Figure A.3: PatchTST/ETTm2/M/192: Hyperparameters sensitivity

columns: constant learning rate and different anneal epochs.
rows: different learning rates and constant anneal epochs.

Bibliography

[1] Dietterich, Thomas G. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[2] Kohavi, Ron, Wolpert, David H, et al. Bias plus variance decomposition for zero-one loss functions. In *ICML*, volume 96, pages 275–283, 1996.

[3] Wolpert, David H. On bias plus variance. *Neural Computation*, 9(6): 1211–1243, 1997.

[4] Breiman, Leo. Random forests. *Machine learning*, 45:5–32, 2001.

[5] Kleinberg, Eugene M. Stochastic discrimination. *Annals of Mathematics and Artificial intelligence*, 1:207–239, 1990.

[6] Bartlett, Peter, Freund, Yoav, Lee, Wee Sun, and Schapire, Robert E. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686, 1998.

[7] Pisetta, Vincent. *New Insights into Decision Trees Ensembles.* PhD thesis, Lyon 2, 2012.

[8] Chen, Tianqi and Guestrin, Carlos. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[9] Kaggle survey 2020. `www.kaggle.com/kaggle-survey-2020`, 2020.

[10] Kaggle survey 2021. `www.kaggle.com/kaggle-survey-2021`, 2021.

[11] Ozcift, Akin and Gulten, Arif. Classifier ensemble construction with rotation forest to improve medical diagnosis performance of machine learning algorithms. *Computer methods and programs in biomedicine*, 104(3):443–451, 2011.

[12] Xie, Yalong, Li, Aiping, Gao, Liqun, and Liu, Ziniu. A heterogeneous ensemble learning model based on data distribution for credit card fraud detection. *Wireless Communications and Mobile Computing*, 2021:1–13, 2021.

[13] Jain, Praphula Kumar, Pamula, Rajendra, and Srivastava, Gautam. A systematic literature review on machine learning applications for consumer sentiment analysis using online reviews. *Computer science review*, 41:100413, 2021.

[14] Pashaei Barbin, Javad, Yousefi, Saleh, and Masoumi, Behrooz. Efficient service recommendation using ensemble learning in the internet of things (iot). *Journal of Ambient Intelligence and Humanized Computing*, 11(3): 1339–1350, 2020.

[15] Alqurashi, Tahani and Wang, Wenjia. Clustering ensemble method.

International Journal of Machine Learning and Cybernetics, 10:1227–1246, 2019.

[16] Song, Yanjie, Suganthan, Ponnuthurai Nagaratnam, Pedrycz, Witold, Ou, Junwei, He, Yongming, Chen, Yingwu, and Wu, Yutong. Ensemble reinforcement learning: A survey. *Applied Soft Computing*, page 110975, 2023.

[17] Ahmed, Nesreen, Atiya, Amir, Gayar, Neamat, and El-Shishiny, Hisham. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29:594–621, 08 2010. doi: 10.1080/07474938.2010.481556.

[18] Ju, Cheng, Bibaut, Aurélien, and van der Laan, Mark. The relative performance of ensemble methods with deep convolutional neural networks for image classification. *Journal of Applied Statistics*, 45(15):2800–2818, 2018.

[19] Shao, Haidong, Jiang, Hongkai, Lin, Ying, and Li, Xingqiu. A novel method for intelligent fault diagnosis of rolling bearings using ensemble deep auto-encoders. *Mechanical Systems and Signal Processing*, 102: 278–297, 2018.

[20] Nie, Yuqi, Nguyen, Nam H, Sinthong, Phanwadee, and Kalagnanam, Jayant. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

[21] Zhou, Tian, Ma, Ziqing, Wen, Qingsong, Wang, Xue, Sun, Liang, and Jin, Rong. Fedformer: Frequency enhanced decomposed transformer for

long-term series forecasting. In *International Conference on Machine Learning*, pages 27268–27286. PMLR, 2022.

[22] Wu, Haixu, Xu, Jiehui, Wang, Jianmin, and Long, Mingsheng. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.

[23] Krogh, Anders and Vedelsby, Jesper. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, 7, 1994.

[24] Chen, Tianqi and Guestrin, Carlos. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[25] Popov, Sergei, Morozov, Stanislav, and Babenko, Artem. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*, 2019.

[26] Wortsman, Mitchell, Ilharco, Gabriel, Gadre, Samir Ya, Roelofs, Rebecca, Gontijo-Lopes, Raphael, Morcos, Ari S, Namkoong, Hongseok, Farhadi, Ali, Carmon, Yair, Kornblith, Simon, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pages 23965–23998. PMLR, 2022.

[27] Ovadia, Yaniv, Fertig, Emily, Ren, Jie, Nado, Zachary, Sculley, David, Nowozin, Sebastian, Dillon, Joshua, Lakshminarayanan, Balaji, and

Snoek, Jasper. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems*, 32, 2019.

[28] Efron, Bradley and Tibshirani, Robert J. *An introduction to the bootstrap*. CRC press, 1994.

[29] Freund, Yoav, Schapire, Robert, and Abe, Naoki. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14 (771-780):1612, 1999.

[30] Hinne, Max, Gronau, Quentin F, van den Bergh, Don, and Wagenmakers, Eric-Jan. A conceptual introduction to bayesian model averaging. *Advances in Methods and Practices in Psychological Science*, 3(2):200–215, 2020.

[31] Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[32] Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15 (1):1929–1958, 2014.

[33] Wan, Li, Zeiler, Matthew, Zhang, Sixin, Le Cun, Yann, and Fergus, Rob. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

[34] Huang, Gao, Sun, Yu, Liu, Zhuang, Sedra, Daniel, and Weinberger,

Kilian Q. Deep networks with stochastic depth. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 646–661. Springer, 2016.

[35] Izmailov, Pavel, Podoprikhin, Dmitrii, Garipov, Timur, Vetrov, Dmitry, and Wilson, Andrew Gordon. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[36] Van der Laan, Mark J, Polley, Eric C, and Hubbard, Alan E. Super learner. *Statistical applications in genetics and molecular biology*, 6(1), 2007.

[37] Hasson, Hilaf, Maddix, Danielle C, Wang, Bernie, Gupta, Gaurav, and Park, Youngsuk. Theoretical guarantees of learning ensembling strategies with applications to time series forecasting. In *International Conference on Machine Learning*, pages 12616–12632. PMLR, 2023.

[38] Chen, Jie, Zeng, Guo-Qiang, Zhou, Wuneng, Du, Wei, and Lu, Kang-Di. Wind speed forecasting using nonlinear-learning ensemble of deep learning time series prediction and extremal optimization. *Energy conversion and management*, 165:681–695, 2018.

[39] Qiu, Xueheng, Zhang, Le, Ren, Ye, Suganthan, Ponnuthurai N, and Amaratunga, Gehan. Ensemble deep learning for regression and time series forecasting. In *2014 IEEE symposium on computational intelligence in ensemble learning (CIEL)*, pages 1–6. IEEE, 2014.

[40] Gheyas, Iffat A and Smith, Leslie S. A novel neural network ensemble architecture for time series forecasting. *Neurocomputing*, 74(18):3855–3864, 2011.

[41] Lin, Weixuan and 0044, Di Wu. Residential electric load forecasting via attentive transfer of graph neural networks. In *IJCAI*, pages 2716–2722, 2021.

[42] Saadallah, Amal, Tavakol, Maryam, and Morik, Katharina. An actor-critic ensemble aggregation model for time-series forecasting. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2255–2260. IEEE, 2021.

[43] Bayati, Abdolkhalegh, Nguyen, Kim-Khoa, and Cheriet, Mohamed. Gaussian process regression ensemble model for network traffic prediction. *IEEE Access*, 8:176540–176554, 2020.

[44] Jiajun, He, Chuanjin, Yu, Yongle, Li, and Huoyue, Xiang. Ultra-short term wind prediction with wavelet transform, deep belief network and ensemble learning. *Energy Conversion and Management*, 205:112418, 2020.

[45] Qiu, Xueheng, Ren, Ye, Suganthan, Ponnuthurai Nagaratnam, and Amaratunga, Gehan AJ. Empirical mode decomposition based ensemble deep learning for load demand time series forecasting. *Applied soft computing*, 54:246–255, 2017.

[46] Fan, Chaodong, Ding, Changkun, Zheng, Jinhua, Xiao, Leyi, and Ai, Zhaoyang. Empirical mode decomposition based multi-objective deep

belief network for short-term power load forecasting. *Neurocomputing*, 388:110–123, 2020.

[47] Lin, Ruibin, Lv, Xing, Hu, Huanling, Ling, Liwen, Yu, Zehui, and Zhang, Dabin. Dual-stage ensemble approach using online knowledge distillation for forecasting carbon emissions in the electric power industry. *Data Science and Management*, 6(4):227–238, 2023.

[48] Floratos, Pavlos, Tsantekidis, Avraam, Passalis, Nikolaos, and Tefas, Anastasios. Online knowledge distillation for financial timeseries forecasting. In *2022 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, pages 1–6. IEEE, 2022.

[49] Campos, David, Zhang, Miao, Yang, Bin, Kieu, Tung, Guo, Chenjuan, and Jensen, Christian S. Lightts: Lightweight time series classification with adaptive ensemble distillation. *Proceedings of the ACM on Management of Data*, 1(2):1–27, 2023.

[50] Sloughter, J McLean, Gneiting, Tilmann, and Raftery, Adrian E. Probabilistic wind speed forecasting using ensembles and bayesian model averaging. *Journal of the american statistical association*, 105(489):25–35, 2010.

[51] Du, Liang, Gao, Ruobin, Suganthan, Ponnuthurai Nagaratnam, and Wang, David ZW. Bayesian optimization based dynamic ensemble for time series forecasting. *Information Sciences*, 591:155–175, 2022.

[52] Saadallah, Amal, Jakobs, Matthias, and Morik, Katharina. Explainable

online ensemble of deep neural network pruning for time series forecasting. *Machine Learning*, 111(9):3459–3487, 2022.

[53] Zhou, Yunyi, Chu, Zhixuan, Ruan, Yijia, Jin, Ge, Huang, Yuchen, and Li, Sheng. ptse: A multi-model ensemble method for probabilistic time series forecasting. *arXiv preprint arXiv:2305.11304*, 2023.

[54] Keskar, Nitish Shirish, Mudigere, Dheevatsa, Nocedal, Jorge, Smelyanskiy, Mikhail, and Tang, Ping Tak Peter. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[55] Huang, Gao, Li, Yixuan, Pleiss, Geoff, Liu, Zhuang, Hopcroft, John E, and Weinberger, Kilian Q. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

[56] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, and Jones, Llion. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[57] Garipov, Timur, Izmailov, Pavel, Podoprikhin, Dmitrii, Vetrov, Dmitry P, and Wilson, Andrew G. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, 31, 2018.

[58] Ruppert, David. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

[59] Polyak, Boris T and Juditsky, Anatoli B. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

[60] `https://pytorch.org/blog/pytorch-1.6-now-includes-stochastic-weight-averaging/`, 2020.

[61] Stephan, Mandt, Hoffman, Matthew D, Blei, David M, et al. Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research*, 18(134):1–35, 2017.

[62] Maddox, Wesley J, Izmailov, Pavel, Garipov, Timur, Vetrov, Dmitry P, and Wilson, Andrew Gordon. A simple baseline for bayesian uncertainty in deep learning. *Advances in neural information processing systems*, 32, 2019.

[63] Izmailov, Pavel, Maddox, Wesley J, Kirichenko, Polina, Garipov, Timur, Vetrov, Dmitry, and Wilson, Andrew Gordon. Subspace inference for bayesian deep learning. In *Uncertainty in Artificial Intelligence*, pages 1169–1179. PMLR, 2020.

[64] Wilson, Andrew G and Izmailov, Pavel. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.

[65] Yang, Guandao, Zhang, Tianyi, Kirichenko, Polina, Bai, Junwen, Wilson, Andrew Gordon, and De Sa, Chris. Swalp: Stochastic weight averaging in low precision training. In *International Conference on Machine Learning*, pages 7015–7024. PMLR, 2019.

[66] Gupta, Vipul, Serrano, Santiago Akle, and DeCoste, Dennis. Stochastic weight averaging in parallel: Large-batch training that generalizes well. *arXiv preprint arXiv:2001.02312*, 2020.

[67] Laine, Samuli and Aila, Timo. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

[68] Tarvainen, Antti and Valpola, Harri. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *Advances in neural information processing systems*, 30, 2017.

[69] Athiwaratkun, Ben, Finzi, Marc, Izmailov, Pavel, and Wilson, Andrew Gordon. There are many consistent explanations of unlabeled data: Why you should average. *arXiv preprint arXiv:1806.05594*, 2018.

[70] Nikishin, Evgenii, Izmailov, Pavel, Athiwaratkun, Ben, Podoprikhin, Dmitrii, Garipov, Timur, Shvechikov, Pavel, Vetrov, Dmitry, and Wilson, Andrew Gordon. Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in artificial intelligence workshop on uncertainty in Deep learning*, 2018.

[71] Fort, Stanislav and Jastrzebski, Stanislaw. Large scale structure of neural network loss landscapes. *Advances in Neural Information Processing Systems*, 32, 2019.

תוכן עניינים

שלמי תודה

תודה עמוקה למנחים שלי, ד"ר יקיר ברנצ'קו וד"ר עמרי אייזנקוט, על האמצעים שהעמידו לרשותי, על ההדרכה הבהירה וגם על החופש והעצמאות ועל התמיכה. היה זה כבוד לעבוד עם שני חוקרים גדולים מתחומי ידע שונים של מדעי הנתונים.

מילות מפתח

למידת אנסמבל, מיצוע משקלים, סדרות עתיות, חיזוי ארוך־טווח, מסלול מיטוב, משטח פונקציית ההפסד.

אוניברסיטת בן־גוריון בנגב

עבודת גמר לתואר מוסמך למדעי ההנדסה (M.Sc.)

יובל סימן־טוב לוי

ניסן התשפ"ד אפריל 2024

מיצוע משקלים באימון רשתות לחיזוי ארוך־טווח של סדרות עתיות

תקציר

הפריחה האחרונה של מודלי חיזוי מזינה את המשך הרדיפה הבלתי־פוסקת אחר שיפורים נוספים בארכיטקטורות הקיימות. חלופה בולטת למעגל זה היא למידת אנסמבל, הידועה ביכולתה לשפר ביצועים ביחס לכל אחד ממודלי הבסיס. אכן, גישה זו יושמה בהצלחה גם לחיזוי של סדרות עתיות באמצעות מגוון שיטות. ברם המשותפת לכולן, היא התקורה החישובית הגבוהה הנלוות להן, אשר מונעת שימוש נפוץ יותר. במקום יצירת אנסמבל מפורש של רשתות עמוקות, באנסמבל מרומז, רשת עמוקה אחת לומדת באופן כזה שהיא מתנהגת כמו אנסמבל שלם, ללא עלות חישובית נוספת. מיצוע המשקולות הנלמדות במהלך האימון היא שיטה עם מטרה דומה, אשר זכתה למאמרי־המשך רבים וליישום במגוון משימות רחב. אולם למיטב ידעתנו, טרם בסדרות עתיות. עבודה זו בודקת לראשונה את השפעת הליך זה על ביצועי שלוש רשתות עמוקות שונות עבור חיזוי ארוך־טווח.

אוניברסיטת בן־גוריון בנגב

הפקולטה למדעי ההנדסה

המחלקה להנדסת תעשייה וניהול

מיצוע משקלים באימון רשתות לחיזוי ארוך־טווח של סדרות עתיות

חיבור זה מהווה חלק מהדרישות לקבלת התואר מוסמך למדעי ההנדסה(M.Sc.)

מאת: **יובל סימן־טוב לוי**

בהנחיית: **ד"ר יקיר ברנצ'קו וד"ר עמרי אייזנקוט**

חתימת המחבר: [illegible] תאריך: 17/04/2024

אישור המנחים: [illegible] [illegible] תאריך: 17/04/2024

אישור יו"ר ועדת תואר שני מחלקתית:

[illegible] תאריך: 17/04/2024

ניסן התשפ"ד **אפריל** 2024

אוניברסיטת בן־גוריון בנגב

הפקולטה למדעי ההנדסה

המחלקה להנדסת תעשייה וניהול

מיצוע משקלים באימון רשתות לחיזוי ארוך־טווח של סדרות עתיות

חיבור זה מהווה חלק מהדרישות לקבלת התואר מוסמך למדעי ההנדסה (M.Sc.)

מאת: **יובל סימן־טוב לוי**

ניסן התשפ"ד **אפריל** 2024