Multi-step Forecasting via Multi-task Learning

Shayan Jawed, Ahmed Rashed, Lars Schmidt-Thieme
Information Systems and Machine Learning Lab
University of Hildesheim
Hildesheim, Germany
{shayan,ahmedrashed,schmidt-thieme}@ismll.uni-hildesheim.de

***Abstract*—Multi-task learning is an established approach for improving the generalization of a model. We explore multi-task learning in the context of time series forecasting. Specifically, we look into a multivariate setting where main and auxiliary series are to be forecasted for multi-step ahead. This results in an interesting multi-task learning problem formulation where the learning tasks come from future horizon of main and auxiliary series both. Our proposed method relies firstly on enumerating multiple Convolutional network architectures to balance the number of shared and non-shared layers between different time series tasks. Also, as multi-step strategies minimize forecast errors over the complete horizon, loss functions would be at different scales based on model uncertainty for near versus distant future. For this reason we propose a factorization of the weight vector for the learning tasks with respect to their categorization of belonging to main or auxiliary series and index in future. An optimal number of shared and non-shared layers together with a novel weighted loss, results in superior performance over 2 real-world datasets compared with several baselines.**

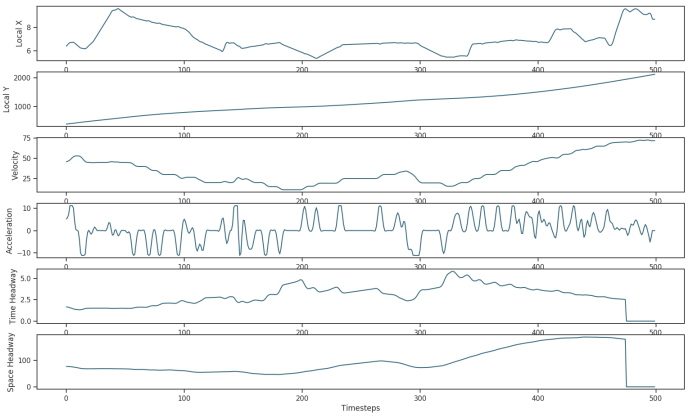


Fig. 1: Example of a multivariate time series from the vehicle trajectory estimation dataset; Local X and Y are considered target series, the rest could be modeled as auxiliary.

I. Introduction

Reliable time series forecasts are required for planning and goal setting in many areas of scientific and business activities. However, producing high-quality forecasts is not an easy task even for the most skilled analysts. The difficulty only increases where in real life scenarios, a forecaster encounters multivariate time series data. In such cases, useful information in related tasks could be leveraged to increase model capacity. Also, a common scenario might require multi-step ahead forecasts for varying future horizons.

Forecasting multiple steps ahead is a more challenging problem where points in distant future would be comparatively harder to forecast for than ones in near future. The inherent structure of a time series also gives rise to the complex dependence between observations recorded at successive time steps. There are three main approaches for generating multi-step predictions, namely recursive, direct and joint strategies. In a recursive strategy, a model predicts for a single step and uses the prediction as input to iteratively predict ahead whereas in a direct strategy, a separate model is built to predict for each time step. The focus of this paper is however on the joint strategy where the model forecasts the whole horizon all at once in a multi-task manner.

Multi-task learning is an important paradigm in machine learning which builds upon the idea of sharing knowledge between different tasks [1]. A set of tasks is learned in parallel, aiming to improve performance over each task compared with learning one of these tasks in isolation. The approach has been tried and tested across a broad domain of problems stemming from, for example natural language processing [2] and computer vision [3]. A multi-task learning problem can also be formulated with respect to main and auxiliary tasks. Auxiliary tasks are motivated by the intuition that for most problem settings, performance over a single task is of primary importance. However, in order to still reap the benefits of multi-task learning, related tasks could be modeled as auxiliary tasks. These exist solely for the purpose of learning an enriched hidden representation that could lead to a lift in prediction accuracy over the main tasks. Reference [1], one of the foremost papers published with respect to multi-task learning defined tasks predicting for different characteristics of the road as auxiliary tasks whereas the main task was to predict the steering direction in a self-driving vehicle. The intuition of using multi-task learning for multivariate time series domain fits naturally as both the tasks themselves are correlated and also the internal representations that could be used for the different tasks are in turn correlated as well [1].

In this paper, we exploit multivariate time series by formulating main tasks to be the future points needed to be forecasted for a target series. In contrast, we adopt the other series's points as auxiliary tasks. This results in an interesting multi-task problem formulation because of the rich output space to be modeled. However, making the problem challenging at the same time. Specifically the questions that arise are,

first, how much of a hidden representation should be shared between the different tasks? Second, how to model an output space which has a strong bias towards the main tasks? Third, how to cater for the difference in uncertainty involved with the prediction of near vs distant future?

We conduct an extensive empirical study to investigate the first question. We exhaustively search for the optimal combination of shared and non-shared layers between the different time series tasks modeled via a Convolutional Neural Network (CNN). The other two questions can be answered by adopting task weights. Specifically, we can model such that in general, tasks from the target series are weighted more than the ones from auxiliary series. Moreover, the tasks of a series would have to be weighted in a decaying scheme to take into account the model's uncertainty with regard to distant vs near future. However, searching for each task's weight is not feasible because of computational complexity. The number of weights could explode drastically if we wish to predict for a decent sized horizon in standard multivariate setting.

We propose a novel scheme to tackle this expansion in the number of weights. Precisely, a factorization of the weight vector for the learning tasks based on the categorization of tasks into main or auxiliary tasks and their index in future. Our factorization leads to searching for only two parameters that cater for such a weighting scheme described above. To the best of our knowledge this is the first work that aims to weight this many tasks. Also, prior multi-task learning approaches have adopted either uniform weights or manually tuned a handful of weights. Examples for such include [4], [5].

In summary, our core contributions can be listed as follows:

- Training a variety of Split network architectures [6] for time series forecasting in order to search for a right balance between task specific and hidden representations in context of target and auxiliary series.
- We propose a novel multi-task loss that tackles the challenging problem of modeling loss functions at different scales and has a strong bias towards main tasks.
- Demonstrating the importance of loss weights for number of tasks at a significantly higher order of magnitude notably than reported previously.
- A thorough ablation study that is built up by rigorously testing the effect of separate building components of proposed method. Ultimately, proving the method on whole is well founded.

II. Related Work

In this section we sketch an overview of the related work in multi-step forecasting. Secondly, we refer notable works belonging to multi-task learning. An extensive survey on multi-step ahead time series forecasting can be found at [7]. Similarly, for multi-task learning at [8].

We firstly note works in recursive strategy such as [9] where iterative forecasts were generated using Support Vector Machine Regression (SVR) and benchmarked on artificial time series data. The authors of [10] did multi-step prediction of high resolution wave power via Recurrent Neural Networks (RNNs). Although powerful, Long Short Term Memory (LSTMs) have mostly replaced RNNs by effectively learning longer term dependencies. Multi-step predictions were produced via recursively forecasting in LSTM based encoder-decoder architectures such as in [11] which forecasted vehicle trajectories. A related point is to consider machine translation works such as [12] where predictions were also generated recursively.

We can also note works that fall under the direct strategy. A multivariate ARIMA direct model for predicting economic processes was discussed in [13]. Non-linear machine learning models include [14] where an approach to predict the operating conditions of machines based on decision trees in conjunction with direct strategy was proposed. Moreover, neural networks and K-Nearest Neighbors have also been used for multi-step prediction with direct strategy [15].

We note comparison between the two strategies in [16]. However, we cannot conclude in essence which strategy is optimal, as it depends on the data generating process, the time series length, the model complexity and the forecast horizon. Therefore, often an empirical study is needed. It is also worth noting that both recursive and direct techniques for multiple-step forecasting share a drawback in that they model from data a single output. Variable y_{t+1} in the recursive case and the variable y_{t+k} in the direct case respectively. For long term prediction, both neglect the existence of stochastic dependencies between future values for example y_{t+k} and y_{t+k+1} [17]. Subsequently, multi-output strategies (also known as Joint strategy) [1], [18] were proposed that avoid the naive conditional independence assumption in the direct strategy.
Notable works for aforementioned joint strategy include [15], [18] where neural networks were trained with all three strategies and the joint strategy was shown to perform the best. The same insight could be drawn from [19] where temporal modeling was done for vehicle trajectory estimation for multi-step ahead with Convolutional Neural Networks trained with joint strategy. Neural networks can naturally deal with multiple outputs however for other models it is not as straightforward. However, we note the extensions built up on the traditional methods for the task of forecasting multiple steps [20] for support vector machines and [21] for local modeling approaches both encouraging utilizing the joint strategy. Still, neural networks remain the choice for multiple step forecasting with the joint strategy. An extensive review [7] where all the aforementioned strategies were compared on a large scale experimental benchmark, the joint strategy was shown to achieve better predictions.

On the other hand, we review the notable works published with respect to Multi-task learning. Previously noted works that utilized the joint strategy fall under the umbrella of multi-task learning. With regard to the question motivated in the introduction about sharing the hidden representation we note two design choices. Hard-parameter sharing has been a recurring theme in most multi-task learning works. This design choice is based on sharing all the hidden layers between all the tasks with the exception of final layers being task specific.

In contrast, soft-parameter sharing models each task with

its own model and parameters. In order to incorporate multi-task learning, regularization schemes are adopted which try to minimize the distance between the parameters such as [22]. Another scheme, *Split networks* shown in [6], falls under the soft-parameter sharing domain however, instead of using a regularization to force the network to have shared parameters, the complete network is splitted in shared and non-shared layers. We focus on this scheme in the paper.

With regard to related works that adopted auxiliary tasks we note in addition to [1] works such as [23] where a multi-task learning problem was formulated for facial landmark detection task. It was shown that by adopting the head pose estimation and facial attribute inference as correlated tasks the performance on main task of detection went up. One of the most prominent works in object detection [24] is based on a multi-task loss. The loss is a combined classification and regression loss for probability distribution and then bounding box regression offsets for the region. In speech synthesis, [25] used a single architecture to jointly predict the phoneme duration and time-dependent fundamental frequency which depend on each another. Modeling the two related tasks together follows natural intuition as one depends on another. Lastly, we note that only a few works have incorporated multi-task learning with regard to time series data. We note [4] predictions were generated for clinical tasks using patient time series data. The authors proposed a heterogeneous multi-task LSTM architecture to jointly model these clinical prediction problems. Interestingly, a custom loss function was used that was formed by weighting the individual loss functions and catered for reasonable progress on all four learning tasks. Recent work by [5], proposes a problem formulation with respect to a target time series like ours. They propose an Auto Encoder Convolutional Recurrent Neural Network that firstly learns filters from doing convolutions across the separate time series including the target series. Later, after applying pooling to each of the series output, they are merged and fed to an RNN prediction layer that outputs multi-step prediction for target time series. The Auto Encoder however simultaneously reconstructs all input time series making the model multi-task.

Our work is similar to this, however we try to model the problem of multi-step predictions of target series by exploiting the shared representation between convolutional layers applied on different channels of multivariate time series. More importantly, unlike the previous works that have used uniform or manually tuned a handful of weights we propose a novel multi-task loss that caters for a significantly higher number of tasks by their index in future and while simultaneously balancing the relation between main and auxiliary series targets.

III. Method

We formulate the problem as a multi-variate, multi-step time series regression process, where the objective is to predict the future values of N time series, $Y = \{Y_1, Y_2, ..., Y_N\}$ given the past input $X = \{X_1, X_2, .., X_N\}$. We assume that each of the time series is multivariate with K channels. The i^{th} time series can be further enumerated as K separate time series, $Y_i = Y_i^*, Y_i^1, Y_i^2, ...Y_i^{K-1}$ and similarly $X_i = X_i^*, X_i^1, X_i^2, ..., X_i^{K-1}$. We denote by Y_i^* the target time series, and the others as auxiliary. The past values of the i^{th} target time series are defined as $X_i^* = \{x_{i,t=0}^*, x_{i,t=1}^*, .., x_{i,t=h}^*\}$, and the forecast values are defined as $Y_i^* = \{y_{i,t=h+1}^*, y_{i,t=h+2}^*, .., y_{i,t=H}^*\}$. Where h and H represent the maximum number of time steps for input and output respectively. Further, we denote the total time series length with $T = h + \delta H$ where δ caters for the case where H is at a lower frequency than h.

A. Multi-step forecasting strategies

We now provide a formal introduction to the three implementation choices for multi-step prediction of a target series $Y^* = \{y_{t=0}^*, y_{t=1}^*, ..., y_{t=H}^*\}$ [15], [19].

1) Iterative strategy: Model training occurs with only a single step prediction which is then recursively fed to the model to forecast ahead. To fix ideas, we can express the prediction for a single step ahead:

$$Y_{t=h+1}^* = f(X^*) \tag{1}$$

Similarly, we can write the equation for two time steps ahead:

$$Y_{t=h+2}^* = f(X^* \cup Y_{t=h+1}^*) \tag{2}$$

The process is followed until $t = H$. An obvious issue with this approach is that errors made early on can significantly alter the predictions for the later timesteps.

2) Direct strategy: The direct strategy addresses the multi-step forecasting problem by utilizing multiple independent models each catering for one of the disjoint subset of a horizon: If we consider two such horizons:

$$\begin{aligned} Y_{t=h+1}^* &= f(X^*) \\ Y_{t=h+2}^* &= g(X^*) \end{aligned} \tag{3}$$

The advantages of this approach over the former are pronounced wherein values within a horizon vary significantly, as the strategy models each task independently.

3) Joint strategy: The joint method can be characterized by its structured prediction over a complete horizon $t = h + 1...t = H$ all at once. Mathematically,

$$Y^* = f(X^*) \tag{4}$$

The joint strategy can be distinguished from the former approaches by it's complexity and strong parameter sharing among multiple tasks modeling for long range horizons.

B. Split Networks for Multivariate Time Series

In this section we introduce Split Network Architectures from [6], and describe how these can be extended to model shared and non-shared features among target and auxiliary series tasks in a multivariate time series. Given that our tasks are related, like we have established, our motivation to use the split networks is to potentially isolate the uncertainty in one task from the other to some extent. As an example, consider the hard-parameter sharing convolutional neural networks that would try to learn filters whose width is the same number of channels as the dimensionality of multivariate time series,

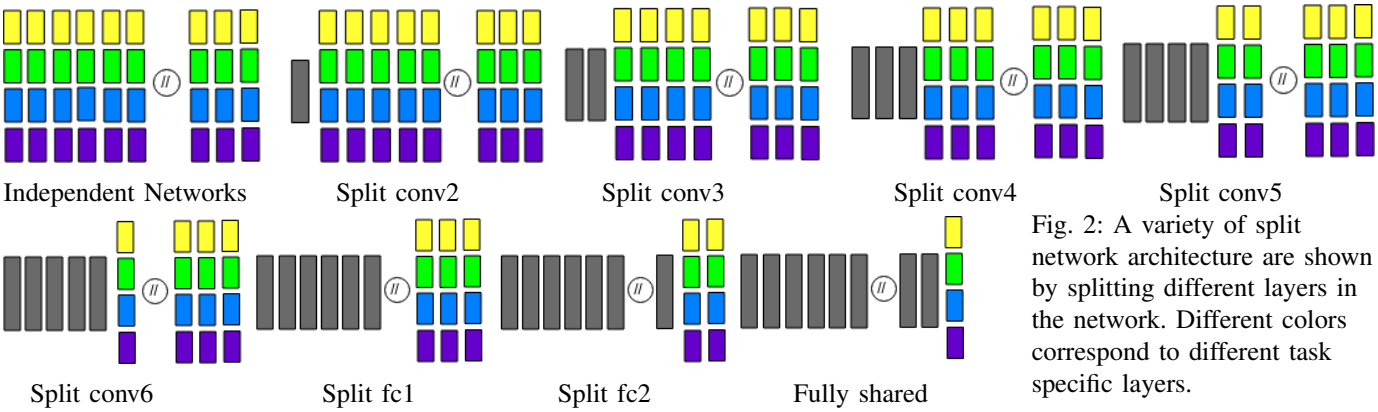


Fig. 2: A variety of split network architecture are shown by splitting different layers in the network. Different colors correspond to different task specific layers.

then some representations might be insightful, such as if there is an increase in the acceleration of vehicle then inherently this would also correspond to an increase in the y-coordinate too, as in Fig. 1. However, this can also have a disadvantage as the acceleration component might have a larger inherent uncertainty which might lead to performance degradation in the outputs of both. If however, the task uncertainty can be controlled in some manner, that is if some channels share the layer up-to some controlled extent then the network's performance might increase. With this in mind, we try to search for the optimal combination of shared and non-shared layers for both the datasets.

The intuition behind split networks is to firstly create separate convolutional layers for each input channel. Then a split is defined which controls the number of shared and non-shared layers to the left and right of the network. Fig. 2, describes the split networks that were tried on the optimal network structure found with hyper-parameter optimization.

Finding the split index is itself a hyper-parameter. An exhaustive search follows, to determine how much of the hidden representation is to be shared across the input channels. The search starts by sharing the first convolutional layer and moves to successively adding more shared layers in the network. The last step denotes the network that has all layers shared across all input channels except the final task specific layers. Thus, split architectures allow for a varying amount of shared and task-specific representation.

Our implementation of the split networks is different from the one in [6], as we are interested in processing time series data and specifically the input channels are not only for whom we wish to learn shared features for but also then predict these channels in future. This can be a problem as the fully connected layers that output for a task need to take into account features learned for each input channel. The split networks proposed in [6], in contrast, have different task specific layers always receive whole image that is all input channels at the same time. We propose to remedy by adopting a concatenation operation that can merge all channel specific features before feeding those to fully connected layers. Although there is no limitation on learning task specific fully connected layers as we do so in split networks Split fc1 and Split fc2 as shown in Fig. 2 albeit now with complete input, the placement of the concatenation operation is justified after the convolutional layers as a design choice.

C. Auxiliary task weights

The primary aim of the paper is to improve multi-step forecast of a target time series by multi-task learning. We wish to explore the multi-task learning paradigm by incorporating auxiliary tasks in the model which forecast for other series in a multivariate time series. This results in an interesting multi-task problem formulation with $K \times H$ tasks to be optimized. A naive approach would be to manually tune each task's weight in the weighted sum [3].

$$L_{total} = \sum_{i} w_i L_i \tag{5}$$

Such an approach has been dominant in previous works [4], [6]. However, it can be impractical for the problem formulation at hand. Searching for each tasks's weight is not computationally feasible as the number of weights could explode drastically in the case of predicting for a decent sized horizon in a standard multivariate setting. In order to tackle this, we exploit the following two insights:

i) Since the aim is to forecast for the complete horizon instead of a single point in future, the loss functions would be scaled proportional to their corresponding timestep's index in future.

ii) If we wish to forecast the auxiliary series for the same number of steps ahead in the future, firstly they will also have loss functions at different scales like above, however, given they are auxiliary targets we would like to weight them less than the main targets.

The reason to weight the auxiliary targets less than the main targets is to penalize the model more on the errors that it makes on the main targets rather than the errors on auxiliary targets. Given that we wish to jointly learn all the tasks, we devise a strategy with respect to the above two insights. Firstly, we define an exponential decaying weighting scheme for the target series denoted with β. Secondly, we generate the weights of auxiliary targets by multiplying the weights generated from this exponential decaying weighting scheme with α, where

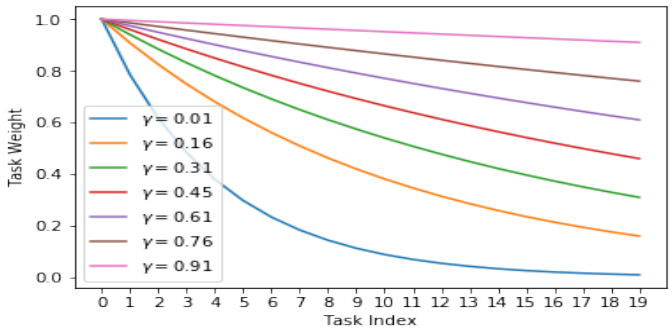


Fig. 3: Task weights generated with different values of γ for a forecasting horizon of 20 tasks.

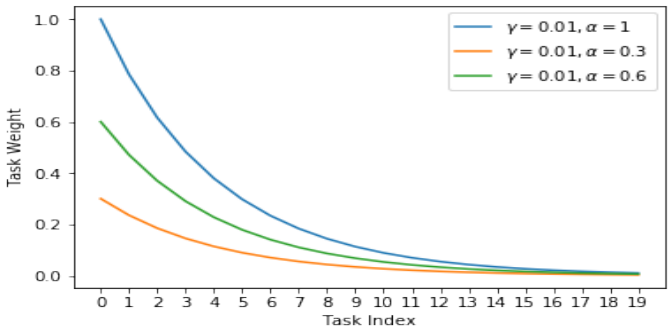


Fig. 4: Scaling the target series weights for two auxiliary series with different values of α.

$\alpha \in [0, 1]$. The loss function for the target series could be stated as follows:

$$L(Y^*, \hat{Y}^*) = \frac{1}{N \times H} \sum_{j}^{H} \sum_{i}^{N} (Y_{ij}^* - \hat{Y}_{ij}^*)^2 \quad (6)$$

A weighted multi-task learning objective formulated with respect to K series could be given as follows:

$$L(Y, \hat{Y}) = \frac{1}{K \times H \times N} \sum_{k}^{K} \sum_{j}^{H} w_{ij} \sum_{i}^{N} (Y_{ij}^k - \hat{Y}_{ij}^k)^2 \quad (7)$$

We propose a novel factorization of the weight vector w_{ij} for the $K \times H$ learning tasks with α and β as follows:

$$L(Y, \hat{Y}) = \frac{1}{K \times H \times N} \sum_{k}^{K} \alpha_k \sum_{j}^{H} \beta_j \sum_{i}^{N} (Y_{ij}^k - \hat{Y}_{ij}^k)^2 \quad (8)$$

Specifically, β_j represents the weight for task to predict the series at timestep j regardless of it being a main or auxiliary task. Fig. 3 shows a set of such weights generated with the exponential weighting scheme described in the following section. What differentiates between the weights of main tasks and the auxiliary tasks is α_k, which denotes the weight for k^{th} series. For target series we set $\alpha = 1$ and for other auxiliary series we search for an optimal value of α. Fig. 4 shows a set of weights generated with exponential weighting and its scaled versions through α.

D. Exponential weighting

We now discuss how to generate the weight vector $\beta_{1:H}$ for all the tasks in horizon H. We define an exponential weighting [26] for multi-step ahead prediction for the time series. The intuition behind defining such a weighting is that it is comparatively harder for the model to predict for points in the far ahead future because of higher uncertainty. As a result the model has a larger error for such predictions. This becomes a problem especially during training with the joint strategy where the model is trained to minimize the error over all the points at the same time. In specific cases the model might diverge. The exponential weighting is defined as follows:

$$\beta_{1:H} = e^{\frac{-|j - center|}{\tau}} \quad (9)$$

Where, j defines the index of the timestep we wish to forecast and $center$ is the parameter defining centre location of the weighting function. τ defines the decay. We fix $center = 0$, and define $\tau = -(H-1)/ln(\gamma)$ where γ is then the fraction of window remaining at the very end, that is the weight for last timestep. Fig. 3 shows the resulting set of weights from different exponential decay rates.

IV. Experiments

This section starts with a description of the datasets followed by an explanation of the evaluation criteria. We compare the proposed model, a Split network trained with a weighting scheme with various single task and multi-task baselines. These experiments are followed by a thorough ablation study in order to evaluate the performance of separate building blocks of the proposed method.

A. Datasets

We evaluate the performance of proposed method on two real-world public time series datasets. Since the data generating processes are completely different, the proposed method's performance can be judged without bias to similar data generating processes.

1) Vehicle Trajectory Estimation Dataset: Our first dataset is the widely cited Next Generation Simulation (NGSIM) dataset [27]. We consider both of its subsets, US 101 and I-80. Amongst other features NGSIM provides, we utilize the x and y coordinates of the front center of the vehicle in a road-relative frame, the velocity, acceleration, time and space headway. There is also information available regarding vehicle and lane identifiers and vehicle types, however we restrict ourselves to using only the dynamic features and ignore such static features altogether. The Local X coordinate measures the distance from left-most section of the highway to vehicle's front center in feet whereas Local Y coordinate measures the distance between entry edge section in the direction of travel and the front center. A summary of dataset characteristics is provided in Table I. We chose not to model time and space headway series as auxiliary targets as a result of feature analysis done with Gradient Boosted Decision Trees [28] and only use these as input.

Time series samples, N'	6,785
Channels, K	6
Sample Length, T	500
Samples after augmentation, N	142,485
Target Series, Y_i^*	Local X, Local Y
Auxiliary Series, Y_i	Velocity, Acceleration
Input window, h	100
Test window, H	20
Train Horizon	$t \leq 300$
Test Horizon	$400 \geq t < 500$

TABLE I: NGSIM Dataset Characteristics

2) News Popularity dataset: In light of analyzing ever increasing social media data, authors in [29] published a dataset which tracks the popularity of different news items on Facebook, Google News and LinkedIn. The news items belong to 4 different topics: Microsoft, Obama, Palestine and Economy. Popularity is defined as the number of shares an item gets on each of the platform. The authors motivate a horizon of two days for tracking the popularity of each news item, with a period being 3 measurements each hour that is 144 timesteps for each news item over each platform.

Time series samples, N'	83,161
Channels, K	3
Sample Length, T	144
Samples after augmentation, N	831,610
Target Series, Y_i^*	Facebook
Auxiliary Series, Y_i^*	Google News, LinkedIn
Input window, h	36
Test window, H	36
Train Horizon	$t \leq 72$
Test Horizon	$108 \geq t < 144$

TABLE II: News popularity dataset characterisitcs

For both the datasets a challenging evaluation criteria is adopted where the model is judged on out of sample predictions. Precisely, the model has not seen data from the test horizon in training stage.

B. Baselines

Our aim in this subsection is to compare the proposed model to representative works from the three multi-step strategies introduced before. We consider the following baselines:

1) Mean Baseline: The mean baseline outputs the average of all historical data for a target time series for as many steps into future, the prediction is needed for.

2) Last Value Baseline: The baseline propagates the last observed value for H timesteps.

3) Joint Strategy Baseline: We compare our proposed method to multi-task Lasso [30]. In multi-task Lasso an l_1/l_2 block norm is defined over $h \times H$ matrix B with the column $\zeta_t \in \mathbb{R}^h$ as the t^{th} column.

$$||B||_{l_1/l_2} := \sum_{t=1}^{h} ||\zeta_t^1, \zeta_t^2, ..., \zeta_t^H||_2 \tag{10}$$

An l_2 norm is applied to each row of B and an l_1 norm across all these blocks. A coordinate descent optimization is done for the following objective:

$$\hat{B} \in \underset{B \in \mathbb{R}^{h \times H}}{\operatorname{argmin}} \{ \frac{1}{2n} \sum_{t=h+1}^{H} ||y_t^* - X\zeta_t||_2^2 + \lambda ||B||_{l_1/l_2} \} \tag{11}$$

4) Direct Strategy Baselines: We consider Random Forests (RF) and Gradient Boosted Decision Trees (XGB) as the direct strategy baselines. We train H many RF and XGB models each, to predict for each timestep ahead. Directly applied to our setting the baselines would require hyperparameter tuning for each target timestep which is impractical. Instead, we resort to default hyperparameters for RF and XGB from implementations in [31] and [28] respectively.

5) Iterative Strategy Baseline: For the iterative strategy baseline, we implement an LSTM based sequence to sequence (Seq2Seq) model where the decoder predicts jointly for all series albeit only 1-step ahead. The encoder and decoder are both LSTM layers with output dimensionality set to 128.

6) 2-D Kernel Baseline: Another baseline is the *Conf.4* explained in the following Section IV-D. The main difference between it and the Split networks is the dimensionality of kernel. The split network will learn 1-D kernels for each of the channels and through soft sharing, the kernel depending on if it belongs to the shared layer will have the same weights for the different channels. In contrast, Conf. 4 will process each channel at the same time with a 2-D kernel. This makes it an interesting baseline to compare the results to.

7) Random Search: Hyperparameter configurations sampled uniformly at random have been shown to be effective for a variety of problems [32]. We compare against model trained with randomly set task weights for loss function in (7) to judge the proposed factorization scheme in (8).

C. Results

This section reports our findings for the proposed method and the baselines for the 3 target series. It is also here that we make the case that although different weighting schemes (7) were adopted for training the model, all results are however stated in terms of an unweighted root mean squared error for fair comparison. All models are judged based on error measure given by squared root of loss in (6) for individual time series. Tables III,IV and V show a comparison of per-step error for the proposed method and baselines for forecasts over the complete horizon H. The results are also stated for the best performing Split network architecture and single task with and without exponential decaying scheme and optimal random weights for target series. A number of interesting observations can be drawn from these results. Firstly, we observe that the proposed method outperforms all other models on the final average error metric in the case of Local Y and Facebook target series. This is only made possible by achieving a consistently low per-step error throughout the forecasting horizon. It follows intuition that errors increase as models predict further ahead as predicting for distant future is harder. Interestingly, this is exactly where the proposed model's strengths lie, which shows a clear advantage of the proposed weighting scheme that can cater for model uncertainty involved in the prediction

of near vs. distant future. The *Rand.* baseline comes second which verifies the intuition of searching for task specific weights. It is worth noting at the same time advantage of setting task specific weights with the proposed factorization which is based on a principled way. We leave the discussion for exploiting rich hidden representations from incorporating auxiliary tasks for the following section and pass over to analyzing how the 3 multi-step prediction strategies compare against each other. We note the recursive strategy baseline *Seq2Seq* performance over the 3 series to be sub-par especially for predictions in distant future which can be expected as errors start accumulating from the first prediction. On the other hand, the baseline *XGB* performs on par with deep networks consistently. In fact, we can note that for the target series of Local X, it's the best performing approach in terms of average error. Its counterpart direct strategy baseline *RF* however fails to model the underlying temporal dynamics. Finally, we note that despite *Lasso* and *Conf. 4* both predict jointly, the proposed method clearly has an edge due to underlying exploitation of rich features together with a principled loss function. Moreover, perhaps surprisingly, a naive baseline such as *Last value*, models an item's popularity better than all other models when it comes to predicting in near future. We infer that this might be because of the way how the evaluation criteria is set up, as the last popularity after 36 hours, might become representative of shares in the next 12 hours if the news item does not go viral.

t	Mean	LV	Lasso	XGB	RF	Seq2Seq	STL	STL(γ)	Conf.4	Spl conv5	Rand.	Prop.
0	3.08	**0.26**	0.48	0.27	0.95	1.14	0.83	0.82	1.56	0.72	0.72	0.71
1	3.13	0.52	0.66	**0.51**	0.62	1.17	0.96	0.96	1.60	0.81	0.80	0.77
2	3.19	0.73	0.83	**0.71**	1.08	1.25	1.08	1.08	1.65	0.89	0.90	0.88
3	3.25	0.91	0.99	**0.90**	1.00	1.34	1.20	1.21	1.72	1.03	1.02	1.00
4	3.32	1.08	1.15	**1.06**	1.13	1.44	1.33	1.34	1.80	1.18	1.17	1.13
5	3.38	1.23	1.29	**1.22**	1.32	1.55	1.45	1.47	1.88	1.28	1.27	1.25
6	3.45	1.37	1.43	1.37	1.42	1.67	1.58	1.59	1.96	1.42	1.40	**1.36**
7	3.51	1.53	1.57	1.51	1.64	1.79	1.70	1.70	2.04	1.54	1.52	**1.48**
8	3.58	1.67	1.70	1.65	1.81	1.90	1.79	1.80	2.11	1.66	1.63	**1.61**
9	3.64	1.80	1.81	1.75	1.95	1.99	1.89	1.89	2.19	1.76	1.74	**1.71**
10	3.71	1.91	1.90	1.84	2.09	2.09	1.98	1.98	2.27	1.85	1.82	**1.80**
11	3.79	2.03	2.00	1.94	2.33	2.18	2.08	2.07	2.35	1.95	1.91	**1.89**
12	3.87	2.17	2.11	2.04	2.36	2.29	2.18	2.17	2.43	2.05	2.01	**2.00**
13	3.96	2.32	2.24	2.17	2.49	2.40	2.29	2.28	2.52	2.17	2.13	**2.13**
14	4.03	2.45	2.34	2.27	2.64	2.49	2.39	2.38	2.62	2.29	2.24	**2.23**
15	4.11	2.57	2.45	2.38	2.75	2.59	2.50	2.48	2.72	2.38	2.34	**2.33**
16	4.18	2.66	2.53	2.47	2.88	2.68	2.58	2.57	2.78	2.47	2.44	**2.42**
17	4.25	2.75	2.61	2.53	3.05	2.76	2.65	2.65	2.85	2.55	2.51	**2.50**
18	4.31	2.82	2.67	2.59	3.20	2.84	2.73	2.73	2.92	2.62	2.57	**2.56**
19	4.35	2.88	2.73	2.64	3.26	2.90	2.78	2.80	2.98	2.68	2.63	**2.62**
μ	3.70	1.79	1.78	**1.70**	2.00	2.02	1.90	1.90	2.24	1.77	1.74	1.71

TABLE III: The proposed method vs. the baselines for all timesteps H, for Local X series. The results (rounded to 2 digits after decimal) are also stated for the different modules of the proposed method.

D. Network Architecture

Neural network architectures vary based on the number of layers and nodes in each layer. We tested different network architectures to model the problem at hand. All architectures are inherently convolutional neural networks with *dilated causal convolutions* [33]. Also, we fixed the kernel sizes to 3 and activations to $ReLU$. However they differ in the specific number of filters, maxpooling, batch normalization and dropout layers. We also catered for network depth. Table

t	Mean	LV	Lasso	XGB	RF	Seq2Seq	STL	STL(γ)	Conf.4	Spl conv5	Rand.	Prop.
0	557.46	13.88	8.00	**4.06**	5.42	10.90	13.33	16.69	24.58	11.90	12.95	10.28
1	573.14	29.82	9.95	**6.55**	7.44	12.24	13.41	17.02	24.22	10.90	11.68	9.57
2	588.92	46.20	13.05	9.55	9.54	14.85	14.06	16.84	23.77	10.27	10.58	**8.82**
3	604.87	62.84	16.61	12.97	12.84	17.90	15.60	18.12	24.18	11.22	11.27	**10.63**
4	620.92	79.53	20.30	16.27	16.95	20.79	16.99	19.13	24.62	12.63	13.08	**11.17**
5	637.11	96.43	24.23	20.34	20.25	24.23	19.22	21.25	26.06	14.50	15.31	**13.88**
6	653.29	113.24	27.99	23.85	24.29	27.33	21.68	23.60	27.13	17.10	16.87	**16.13**
7	669.61	130.13	31.62	27.59	28.43	30.50	24.00	25.84	29.83	18.81	**17.36**	17.74
8	686.11	147.27	35.87	31.74	32.62	34.18	27.20	28.89	32.19	22.90	20.69	20.35
9	702.72	164.53	40.17	35.87	35.58	38.06	30.91	32.53	35.29	25.06	**23.22**	23.88
10	719.29	181.73	44.62	40.53	39.81	42.01	35.19	36.26	38.52	27.16	**26.20**	27.04
11	735.93	199.01	49.05	45.07	44.22	46.08	39.38	40.18	42.02	31.02	**30.32**	30.57
12	752.65	216.34	53.56	48.70	48.53	50.22	43.89	44.03	45.81	33.91	**33.89**	34.02
13	769.35	233.62	57.94	53.04	52.76	54.32	48.04	48.33	49.67	37.55	37.60	**37.45**
14	786.14	250.99	62.35	58.23	56.46	58.49	52.79	52.63	53.61	**40.88**	40.75	41.22
15	802.93	268.34	66.79	61.74	60.17	62.79	57.10	57.06	57.82	**44.53**	44.57	44.82
16	819.66	285.62	71.29	66.96	65.84	67.16	62.27	61.14	62.25	**48.21**	48.47	48.40
17	836.34	302.84	75.74	71.37	68.67	71.64	67.36	65.65	67.16	**51.91**	52.41	52.27
18	853.11	320.15	80.28	76.01	72.92	76.26	72.30	70.33	71.21	**55.75**	56.42	56.18
19	870.04	337.59	84.78	80.85	77.05	80.95	77.67	74.68	75.85	**59.59**	60.50	60.26
μ	711.98	174.00	43.71	39.56	38.99	42.05	37.62	38.51	41.79	29.23	29.20	**28.73**

TABLE IV: Results for the complete horizon of the Local Y series.

t	Mean	LV	Lasso	XGB	RF	Seq2Seq	STL	STL(γ)	Conf.4	Spl fc2	Rand.	Prop.
0	272.90	**10.00**	14.87	22.67	43.09	46.69	60.99	41.07	37.09	27.44	23.41	19.08
1	277.39	**19.16**	22.20	37.10	56.29	47.59	58.90	43.18	38.55	29.87	26.81	22.73
2	279.24	**21.64**	25.12	34.87	49.72	49.20	59.33	43.65	38.41	30.58	27.61	24.49
3	281.32	**24.42**	27.48	41.36	63.81	50.57	59.18	43.98	40.66	31.22	28.57	25.96
4	283.35	**26.62**	29.36	44.79	63.51	51.64	59.42	42.81	42.01	31.85	29.49	28.19
5	285.52	33.97	36.39	46.69	61.06	55.88	62.20	47.30	44.90	37.44	35.49	**33.64**
6	287.33	36.14	38.45	49.33	72.45	57.37	63.10	48.23	44.61	38.61	36.94	**35.13**
7	289.01	37.68	39.87	52.10	66.29	58.79	63.88	48.77	44.84	39.41	37.88	**36.03**
8	291.35	42.34	44.12	54.45	82.09	61.63	65.20	51.59	47.59	42.80	41.58	**39.81**
9	293.07	44.21	46.25	54.25	87.50	62.98	65.01	52.99	47.54	43.48	42.94	**41.08**
10	297.64	52.43	52.53	61.63	84.22	61.55	64.54	58.17	48.40	46.83	**45.08**	46.20
11	301.18	62.65	62.15	64.83	101.35	69.94	71.51	66.72	57.84	55.84	**54.49**	56.07
12	303.61	68.60	67.92	72.37	103.43	75.48	76.39	71.86	63.56	61.68	**60.39**	61.99
13	305.36	69.96	69.34	73.72	110.19	76.54	76.61	73.16	63.54	62.18	**61.09**	62.69
14	308.74	72.33	71.04	77.51	110.72	77.38	78.26	73.72	65.12	63.82	**62.07**	63.11
15	310.66	74.15	72.41	76.98	110.27	79.76	79.41	75.29	66.27	64.92	**63.15**	64.12
16	312.35	76.20	74.26	78.17	111.30	81.75	80.48	76.88	67.54	65.84	**64.43**	65.59
17	315.01	77.78	75.60	86.19	128.23	84.62	80.86	78.24	69.98	67.35	**65.45**	66.82
18	316.62	79.26	76.94	85.84	122.94	86.04	81.04	79.55	70.79	68.35	**66.19**	67.76
19	318.27	80.74	78.22	85.07	125.32	87.40	81.61	80.63	71.29	69.00	**66.77**	68.56
20	319.84	82.75	80.17	86.25	136.15	89.38	82.62	82.11	72.90	70.47	**68.36**	70.29
21	320.98	83.89	81.32	86.60	135.71	90.88	83.56	83.07	73.58	71.77	**69.31**	70.86
22	322.81	85.38	82.54	86.25	134.27	92.31	84.62	84.14	74.72	72.85	73.36	**71.64**
23	324.05	86.44	83.45	88.26	133.39	93.58	85.26	84.88	75.41	73.77	74.51	**72.23**
24	325.62	88.11	85.06	87.71	137.22	95.47	86.43	86.21	76.35	75.28	75.95	**73.46**
25	328.24	92.26	88.94	93.38	144.31	97.85	87.78	89.78	79.36	78.83	79.50	**76.73**
26	329.28	93.10	89.81	96.04	145.25	98.85	88.47	90.42	79.98	79.92	79.97	**77.18**
27	330.69	94.41	90.94	92.66	148.63	100.48	89.28	91.42	80.87	80.84	81.07	**78.07**
28	332.19	95.74	92.07	97.39	145.40	101.59	89.80	92.33	81.70	81.75	81.74	**79.98**
29	333.91	97.38	93.37	94.61	151.01	102.97	91.74	93.35	83.14	82.81	83.05	**80.87**
30	335.66	99.39	95.33	96.58	151.73	104.72	93.08	94.96	85.44	84.10	84.74	**82.31**
31	337.95	101.89	97.58	96.87	150.33	107.30	94.73	97.06	87.87	85.94	87.07	**84.28**
32	346.58	124.99	120.25	120.81	161.68	128.89	118.67	120.09	113.36	111.52	112.93	**110.79**
33	352.86	139.18	134.11	136.13	169.97	142.37	133.11	134.15	128.39	126.43	128.42	**126.40**
34	357.35	145.37	139.75	140.94	176.52	148.09	138.55	139.28	134.40	**131.77**	134.31	131.89
35	359.98	147.89	141.91	144.68	182.98	150.04	139.95	141.07	135.83	**133.60**	136.26	133.60
μ	313.56	74.12	72.80	79.03	115.50	85.21	82.65	77.84	70.38	67.23	66.40	**65.26**

TABLE V: Results for the Facebook series.

VI shows the different network architectures. The network architectures were trained for the 100 epochs on news-popularity dataset with a non-weighted version of multi-task loss shown in (7), where the last three fully connected layers predicted for the last 36 timesteps that is the test set for the popularity of the item on the respective platform. We can see that $Conf.4$ is the leading network architecture. Thus we adopt this architecture in order to build the Split Networks as shown in Fig. 2. It is worth noting that this is the standard protocol to design a network structure in machine learning. As there is no standard network that fits all tasks, one needs to do a systematic experimentation with a robust evaluation protocol for the task at hand. However, that being said, there are some generalizations across a particular data type, like for example deeper networks would be preferred for image classification whereas the same architectures could lead to overfitting on

time series data.

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
	Conv(8)	Conv(8)	Conv(8)	Conv(8)	Conv(24)	Conv(32)	Conv(64)	Conv(64)
	BN	BN	BN		BN	BN	BN	
	Max	Max	Max	Max				
	Conv(16)	Conv(16)	Conv(16)	Conv(16)	Conv(32)	Conv(32)	Conv(64)	Conv(64)
	BN	BN	BN		BN	BN	BN	
	Max	Max	Max	Max				
	Conv(32)	Conv(32)	Conv(32)	Conv(32)	Conv(48)	Conv(64)	Conv(128)	Conv(128)
	BN	BN	BN		BN	BN	BN	
	Max	Max	Max	Max				
	Conv(64)	Conv(64)	Conv(64)	Conv(64)	Conv(64)	Conv(64)	Conv(128)	Conv(128)
	BN	BN	BN		BN	BN	BN	
	Max	Max	Max	Max				
	Conv(128)	Conv(128)	Conv(128)	Conv(128)	Conv(64)			
	BN	BN	BN		BN			
	Max	Max	Max	Max				
	Conv(256)	Conv(256)	Conv(256)	Conv(256)				
	BN	BN	BN					
	Max	Max	GbMax	Max	Max	Max	Max	Max
	Flat	Flat		Flat	Flat	Flat	Flat	Flat
		Drop(15)						
	FC(200)	FC(200)	FC(200)	FC(200)	FC(256)	FC(256)	FC(256)	FC(256)
	FC(100)	FC(100)	FC(100)	FC(100)	FC(128)	FC(128)	FC(128)	FC(128)
					FC(64)	FC(64)	FC(64)	FC(64)
	Drop(15)	Drop(15)			Drop(15)	Drop(15)	Drop(15)	Drop(15)
	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)
	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)
	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)	FC(36)
FB	140.90	109.10	108.41	**72.11**	122.71	134.84	117.40	100.98
GN	3.93	4.22	4.00	**2.14**	4.30	4.30	3.84	2.56
LN	18.35	13.03	14.22	**10.18**	17.28	15.33	15.86	11.64

TABLE VI: Network architectures tested on the news popularity dataset. The results are reported for the test set. We observe that Conf. 4 reports the least error for all three series.

E. Ablation Studies

We seek to answer the questions outlined previously with a thorough ablation study. We firstly isolate the building components of the methodology and test their performance and then gradually piece up the components to compose the proposed method.

Our first set of results in the ablation study is on comparing the performance of single-task networks for the target series with and without exponential decay rates. The reason for doing this is to study the effect of adopting an exponential decaying weighting scheme and have a single-task learning baseline to compare the results to. In order to do so, we take the best performing architecture which we refer to as *Conf. 4* from previous section, and train it with a weighted loss only to predict for the main tasks. We can note the RMSE averaged over the entire horizon for the test set of the three series in Fig. 5. We trained the network with different decay rates, for example $\gamma = 0.01$ corresponds to an exponential decay whereas a much higher value of $\gamma = 0.75$ would correspond to a linear decay. One can observe that adding the different exponential decay rates was unfortunately not fruitful. The averaged results over the horizon did not decrease as shown in Table. VII where for Local Y and Local X, having no exponential decay achieved the lowest error.

Next, we trained split networks based on the architecture from *Conf.4*. For the vehicle trajectory dataset we trained 4 networks, where each predicted for one series tasks. These networks had parameters shared up-to a split. Similarly, for the news popularity dataset, 3 networks were trained. Results are reported in Table. VIII. A number of interesting observations can be made looking into the results. Firstly, we note that having a multi-task objective clearly helped bring the performance up for both the datasets. For the vehicle trajectory dataset we can see that the targets of Local Y have benefited the most where average error over the horizon has gone considerably down, compared to the single task learning objective from Table. VII. It is also worth noting that Local Y tasks are considerably difficult to model than their Local X counterparts, as a vehicle might not change lanes but rather shift gears more frequently. This causes the network to diverge for the Local X tasks although the training loss keeps decreasing. We also state the errors on auxiliary series for the reader's curiosity. On the other hand, it can be observed that the Split architecture *split fc2* has the lowest RMSE for the experiments conducted yet for the Facebook series. The *split fc2* architecture is in-fact sharing all the layers though. However, we also note that the RMSE varies considerably in between the different split architectures. Thus, the intuition of enumerating the possible architectures for the optimal number of shared and non-shared layers was fruitful and we achieve the lowest RMSE yet. Moreover, Fig. 5 shows variability in convergence of top-2 performing split networks for all three target series.

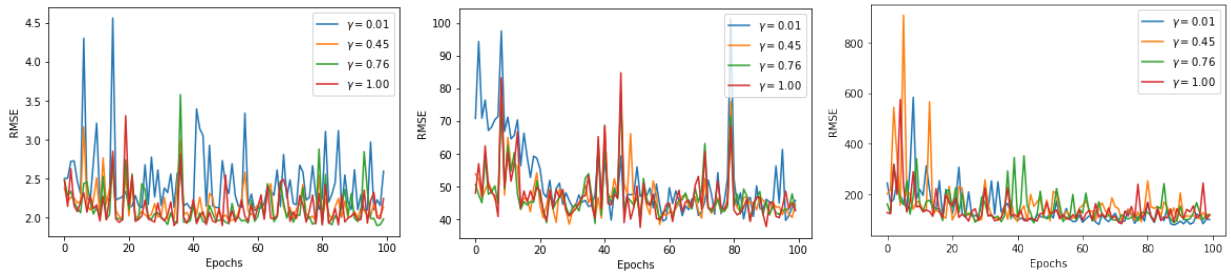
γ	0.01	0.45	0.76	1.00
Local X	2.09	1.95	1.89	**1.89**
Local Y	39.57	38.51	38.86	**37.61**
Facebook	**77.84**	90.39	85.80	82.65

TABLE VII: Averaged results for models trained with and without exponential decay rates over the complete horizon for the three target series. Errors are reported in original units from the dataset, Local X and Local Y in ft. and number of shares for Facebook series.

	Target Series						
Network	Local X	Local Y	Velocity	Acceleration	Facebook	Google	LinkedIn
Indep.	1.80	32.47	4.96	**6.33**	76.80	2.05	9.81
Split conv1	1.81	30.41	4.97	6.60	73.72	**2.02**	9.65
Split conv2	1.80	32.07	4.97	6.45	71.67	2.03	9.62
Split conv3	1.77	32.20	4.97	6.66	77.64	2.32	10.19
Split conv4	1.77	30.32	4.96	6.49	75.12	2.10	9.82
Split conv5	**1.77**	**29.23**	**4.96**	6.65	77.83	2.32	10.62
Split conv6	1.79	29.58	4.97	6.71	72.82	2.17	9.95
Split fc1	1.84	30.63	4.97	6.50	69.33	2.05	9.93
Split fc2	1.88	33.08	4.97	6.43	**67.23**	2.07	**9.73**

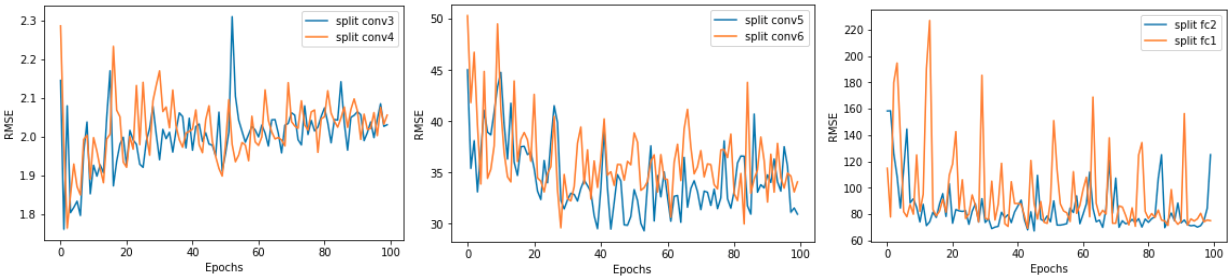
TABLE VIII: We tried different configurations of Split Networks for both the datasets. These networks are characterized by soft-sharing among their parameters up-to the split.

Our next set of results is on studying the effect of incorporating different weights for target and auxiliary series tasks. We select the best-performing split network found for each target series and train it with a weighted multi-task objective (8) at this stage. To incorporate task weights, we employ a random search procedure that samples uniformly at random from the range $(0, 1)$ for the optimal values of γ and α_k. We ran 12 different configurations for both the random



Local X Local Y Facebook

Fig. 5: We plot here the error averaged over the horizon for the three target series from the different exponential weighting schemes. Higher values of γ correspond to linear decays and $\gamma = 1.0$ denotes single task learning without any loss weights.



Local X Local Y Facebook

Fig. 6: A variety of split networks architectures were trained by splitting at different layers with a multi-task objective of generating multivariate multi-step predictions. We plot here for the top-2 performing split networks for the three series. We notice that there is considerable variability in the convergence of split networks which verifies the intuition that parameter sharing depends on task at hand. Also worth noting is how the performance on Local Y increases, but the networks starts diverging for Local X due to optimization of a multi-task loss where Local Y targets dominate the loss.

search baseline and our proposed factorization and for all 3 target series found the minimum error with the proposed factorization. The best hyperparameters were found for Local Y to be ($\gamma = 0.11$, $\alpha_{vel} = 0.82$, $\alpha_{acc} = 0.29$). Similarly, for Local X and Facebook series we found the parameters ($\gamma = 0.70$, $\alpha_{vel} = 0.54$, $\alpha_{acc} = 0.86$) and ($\gamma = 0.05$, $\alpha_{google} = 0.55$, $\alpha_{linkedin} = 0.38$). It is also worth noting that model performance is sensitive to selection of weights.

V. Reproducibility

We first describe the preprocessing of the datasets. Since the task at hand is to strictly model time series, we ignore any type of feature engineering or static information from both of the datasets. We do however, utilize standardization by removing the mean and scaling the features to unit variance as follows. This standardization is applied to each column separately and the mean and standard deviation for each are stored to be used on later inverse transformations. For the vehicle trajectory dataset, we have about 2355 vehicles from I-80 subset and 2490 from the US 101. Almost each vehicle appears in around 3 different tracking sequences. We group by all rows by vehicle ID then by the tracking sequences of 15 minutes each. This can lead to one vehicle providing up to three samples of trajectories. We discard any trajectory sample that is less than 500 points that is less than 50 seconds and end up with 6785 samples. Since for each sample we record the 6 different time-dependent features the input tensor dimension is then $(N \times T \times K)$. Where $N = 6785, T = 500$ and $K = 6$. We shuffle this dataset to remove any correlated trajectories. Next, we augment this data using a sliding window mechanism on the second dimension of the tensor. We define a window size of 100. Each such window covers 10s of past trajectory of the vehicle. We take each window after every 1s that is we do a sliding window operation on the dataset with a stride of 10. We do this operation until $T \leq 300$ as we keep the last sliding window from $T >= 300 \leq T < 400$ for testing. This procedure determines the X and for Y, the targets, we take 20 points each spaced at half a second from the 100 points following the 100 points from X. That is predicting for the last step would translate into predicting the vehicle's position after 10 seconds. By doing the operation 21 times until $T \leq 300$, we end up with a tensor of dimensions $(21 \times 6785 \times 100 \times 6)$. From here, we multiply the first and second dimension to create a tensor of shape $(142485 \times 100 \times 6)$ effectively augmenting the dataset's sample dimension. Similarly, for the other dataset, we ignore any feature engineering except the standardization as was noted earlier. There are around 83,161 items in total from all topics with their popularity tracked for 144 timesteps on 3 platforms. Thereby, generating

a tensor of shape $(83161 \times 144 \times 3)$. We shuffle this tensor to break the correlation between the same topics for example the top 29928 items are from Economy. This should lead the network to generalize well across all items. The sliding window mechanism to augment the data is also utilized for this dataset. With a stride of 4, and a window length of 36 we do 10 passes until $T \leq 108$ which forms the train part of the data. The target for this dataset is to predict for the last 36 timesteps that is the last quarter of the 48 hours timespan. By doing the sliding window, the resulting tensor is of shape $(831610 \times 36 \times 3)$.

VI. Acknowledgements

We gratefully acknowledge the co-funding of our work by Volkswagen Financial Services through Data-driven Mobility Services project.

VII. Conclusion

Our focus in this paper was on the challenging task of multi-step forecasting. We presented multi-task learning methods that could exploit shared information to improve generalization over single-task models. Split networks showed promising results by learning a rich hidden state representation of shared and task specific features. With the intuitions of catering for model uncertainty in distant future and bias towards main tasks set forth, we defined a factorization of the weight vector to derive a principled loss function, which employed by an optimal Split network was shown to outperform a series of baselines. Moreover, the proposed method is broadly applicable to any problem setting involving multivariate input and requiring multi-step forecasting.

References

[1] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[2] X. Liu, J. Gao, X. He, L. Deng, K. Duh, and Y.-Y. Wang, "Representation learning using multi-task deep neural networks for semantic classification and information retrieval," 2015.

[3] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7482–7491.

[4] H. Harutyunyan, H. Khachatrian, D. C. Kale, and A. Galstyan, "Multitask learning and benchmarking with clinical time series data," *arXiv preprint arXiv:1703.07771*, 2017.

[5] R.-G. Cirstea, D.-V. Micu, G.-M. Muresan, C. Guo, and B. Yang, "Correlated time series forecasting using multi-task deep neural networks," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2018, pp. 1527–1530.

[6] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3994–4003.

[7] S. B. Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition," *Expert systems with applications*, vol. 39, no. 8, pp. 7067–7083, 2012.

[8] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[9] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik, "Using support vector machines for time series prediction," *Advances in kernel methods—support vector learning*, pp. 243–254, 1999.

[10] K. Hatalis, P. Pradhan, S. Kishore, R. S. Blum, and A. J. Lamadrid, "Multi-step forecasting of wave power using a nonlinear recurrent neural network," in *2014 IEEE PES General Meeting— Conference & Exposition*. IEEE, 2014, pp. 1–5.

[11] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1468–1476.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[13] G. Chevillon and D. F. Hendry, "Non-parametric direct multi-step estimation for forecasting economic processes," *International Journal of Forecasting*, vol. 21, no. 2, pp. 201–218, 2005.

[14] B.-S. Yang, A. C. C. Tan *et al.*, "Multi-step ahead direct prediction for the machine condition prognosis using regression trees and neuro-fuzzy systems," *Expert systems with applications*, vol. 36, no. 5, pp. 9378–9387, 2009.

[15] D. M. Kline, "Methods for multi-step time series forecasting neural networks," in *Neural networks in business forecasting*. IGI Global, 2004, pp. 226–250.

[16] M. Marcellino, J. H. Stock, and M. W. Watson, "A comparison of direct and iterated multistep ar methods for forecasting macroeconomic time series," *Journal of econometrics*, vol. 135, no. 1-2, pp. 499–526, 2006.

[17] G. Bontempi, S. B. Taieb, and Y.-A. Le Borgne, "Machine learning strategies for time series forecasting," in *European business intelligence summer school*. Springer, 2012, pp. 62–77.

[18] G. Bontempi, "Long term time series prediction with multi-input multi-output local learning," *Proc. 2nd ESTSP*, pp. 145–154, 2008.

[19] S. Jawed, E. Boumaiza, J. Grabocka, and L. Schmidt-Thieme, "Data-driven vehicle trajectory forecasting," *arXiv preprint arXiv:1902.05400*, 2019.

[20] Y. Bao, T. Xiong, and Z. Hu, "Multi-step-ahead time series prediction using multiple-output support vector regression," *Neurocomputing*, vol. 129, pp. 482–493, 2014.

[21] G. Bontempi and S. B. Taieb, "Conditionally dependent strategies for multiple-step-ahead prediction in local learning," *International journal of forecasting*, vol. 27, no. 3, pp. 689–699, 2011.

[22] Y. Yang and T. M. Hospedales, "Trace norm regularised deep multi-task learning," *arXiv preprint arXiv:1606.04038*, 2016.

[23] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, "Facial landmark detection by deep multi-task learning," in *European Conference on Computer Vision*. Springer, 2014, pp. 94–108.

[24] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[25] S. O. Arik, M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman *et al.*, "Deep voice: Real-time neural text-to-speech," *arXiv preprint arXiv:1702.07825*, 2017.

[26] S. Gade and H. Herlufsen, "Use of weighting functions in dft/fft analysis," *B&K Technical Review*, vol. 3, 1987.

[27] J. Colyar and J. Halkias, "Us highway 101 dataset," *Federal Highway Administration (FHWA), Tech. Rep. FHWA-HRT-07-030*, 2007.

[28] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[29] N. Moniz and L. Torgo, "Multi-source social feedback of online news feeds," *arXiv preprint arXiv:1801.07055*, 2018.

[30] G. Obozinski, B. Taskar, and M. I. Jordan, "Joint covariate selection and joint subspace selection for multiple classification problems," *Statistics and Computing*, vol. 20, no. 2, pp. 231–252, 2010.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[32] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[33] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio." *SSW*, vol. 125, 2016.