FORECASTING DEEP LEARNING DYNAMICS WITH APPLICATIONS TO HYPERPARAMETER TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Well-performing deep learning models have enormous impact, but getting them to perform well is complicated, as the model architecture must be chosen and a number of hyperparameters tuned. This requires experimentation, which is time-consuming and costly. We propose to address the problem of hyperparameter tuning by learning to forecast the training behaviour of deep learning architectures. Concretely, we introduce a forecasting model that, given a hyperparameter schedule (e.g., learning rate, weight decay) and a history of training observations (such as loss and accuracy), predicts how the training will continue. Naturally, forecasting is much faster and less expensive than running actual deep learning experiments.

The main question we study is whether the forecasting model is good enough to be of use - can it indeed replace real experiments? We answer this affirmatively in two ways. For one, we show that the forecasted curves are close to real ones. On the practical side, we apply our forecaster to learn hyperparameter tuning policies. We experiment on a version of ResNet on CIFAR10 and on Transformer in a language modeling task. The policies learned using our forecaster match or exceed the ones learned in real experiments and in one case even the default schedules discovered by researchers. We study the learning rate schedules created using the forecaster are find that they are not only effective, but also lead to interesting insights.

1 INTRODUCTION

Machine learning researchers working with deep neural networks spend time looking at training curves and asking themselves the question: How would the results change if I modified some hyperparameters? They run many experiments and, with time, develop a better understanding of how modifications in various hyperparameters affect the learning dynamics of our models.

In this work, we attack this problem of understanding deep learning dynamics with deep learning itself. To this end, we collect a data-set of training curves of deep learning models with a diverse set of hyperparameter schedules. Next, we train an autoregressive deep model to predict the training curves conditioned on the schedule.

More concretely, our model observes the setting of a small number of hyperparameters, such as learning rate, weight decay, or dropout rate. Then, it predicts a small number of values that a researcher would usually look at, such as training and validation loss and accuracy. We predict how these values will change a few hundred training steps later. Then, the model gets new settings for for the next few hundred training steps, and predicts the next values. The model is autoregressive, so it uses the history of its own predictions from previous time-steps to predict the next ones.

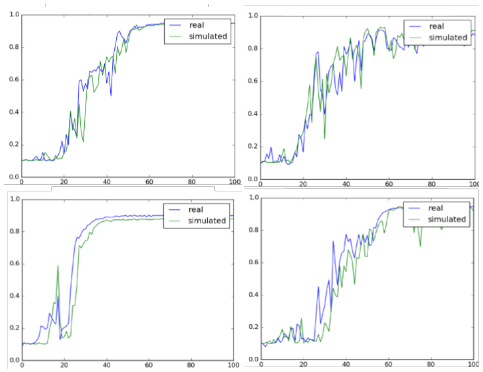

Figure 1: Real and predicted evaluation accuracy curves, see text on the left for details.

We study how to do time-series prediction with autoregressive models even in the presence of stochastic behaviour. We study different losses and introduce a

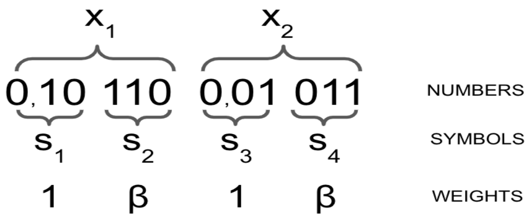

Figure 2: Discretization procedure where x_t is the discretized sequence. It is represented in fixed-precision encoding with 2 base-8 digits per number, so each symbol s_i corresponds to 3 bits of precision. Weights of symbols are decayed with parameter β according to their significance, so the more significant digit has weight 1 and the other has weight β.

loss weight scheme that allows to use Transformer models for stochastic time-series predictions. We show that the new method we introduce outperforms baselines and is able to faithfully approximate the learning dynamics of deep learning models, as shown in the examples in Figure 1.

Having a model of learning dynamics is interesting in its own right, as it allows to study the training process much faster, without running full experiments. But it also opens up the possibility to learn hyperparameter schedules by using reinforcement learning in the model. We experiment with this and validate our forecaster: the policies learned using reinforcement learning and the forecaster work well when training real models. Running the forecaster is of course much faster than running real experiments, so learning a policy in the forecaster is thousands of times faster.

2 Time-Series Modeling with Transformer

Transformer models show very good performance in modeling distributions over discrete sequences, such as text (Vaswani et al., 2017). In this work, we show that they can also be used to predict time series – concretely, training curves of deep learning models. One obvious way to apply Transformer to time-series prediction would be to try to directly predict the next value in the sequence with L2 loss. We show (below) that this is not a good strategy when sequences can be stochastic and introduce a discretization technique that reduces the problem to the standard discrete sequence prediction setting and that yields remarkably good results.

2.1 Discretization

In order to model time series using Transformer, we represent each number in the sequence in fixed-precision, base-k encoding. We concatenate the consecutive base-k digits corresponding to each number, starting from the most significant one. To predict multiple time series at a time, we arrange the representations of a single element of each time series in a fixed order, and then concatenate the representations of each time step, as shown at Figure 2. This way we can model the joint probability distribution of multiple time series using a single model. We train the Transformer decoder to predict consecutive symbols in this discrete sequence autoregressively. We use cross-entropy as the loss function, weighted with exponential decay according to the significance of each digit:

$$L(t,p) = \sum_{i=0}^{n-1} \beta^i H(t_i, p_i)$$

where H denotes cross-entropy, t_i is the deterministic distribution over true digits, p_i is the model prediction and i indexes digits in the fixed precision, base-k encoding of a number, starting from the most significant one.

2.2 Results on synthetic data

To demonstrate the ability of Transformer to predict training curves, we train it on a dataset of synthetically generated curves calculated using the formula:

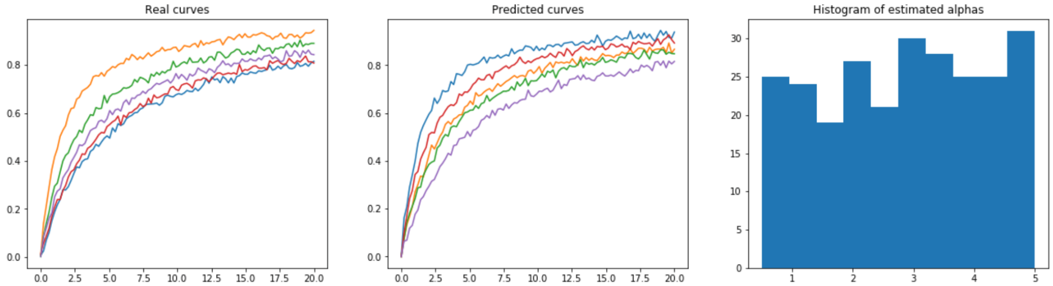

Figure 3: Synthetic curves generated by discrete Transformer decoder with weighted cross-entropy loss together with the histogram of α values. In the generation process we used a uniform distribution and the histogram approximates it well.

$$x_i = 1 - \frac{1}{1 - \frac{i}{\alpha}} + \mathcal{N}(0, \sigma^2)$$

where i is an integer from the interval $[1, N]$ and σ is scale of the noise, set to 0.01. The dataset is designed to mimic training accuracy curves, starting from 0 and converging to 1 in the limit. Rate of the convergence is controlled by the parameter α, sampled uniformly from the interval $[0.5, 5]$ for each curve.

To measure the diversity of the generated curves, we estimate the parameter α from a curve predicted by the model, by averaging over pointwise estimates along the curve:

$$\hat{\alpha} = \frac{1}{N} \sum_{i=0}^{N-1} t(\frac{1}{x_i} - 1)$$

We then visualize the distribution of $\hat{\alpha}$ in a histogram. As shown in Figure 3, Transformer is able to generate diverse and convincingly-looking curves.

As a baseline, we consider modeling the curves without discretization, using the Transformer decoder without embedding and predicting a sequence of real numbers, training the model with L2 loss. During inference, we add Gaussian noise with variance σ^2 to the prediction at each step, to introduce stochasticity. As shown at Figure 4, the curves generated this way tend to collapse to a single shape.

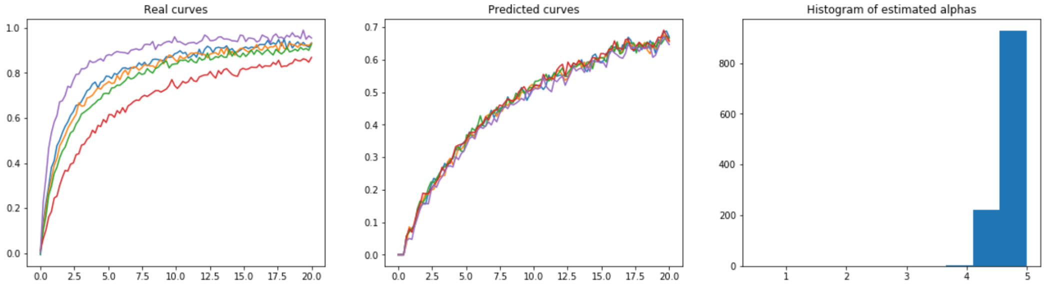

Figure 4: Synthetic curves generated by continuous Transformer decoder with L2 loss together with the histogram of α values that shows collapse.

3 Hyperparameter Tuning using Reinforcement Learning

As is standard in reinforcement learning, we frame the problem of controlling model hyperparameters during training as a partially-observable Markov decision process. Each transition in the MDP corresponds to n steps of training followed by an evaluation on a held-out set of 5% of the training data. Observations are the current values of training and evaluation accuracy and loss. Note that the agent does not have access to full state of the environment, because that would encompass the current values of all parameters of the model, which is intractable as an input for the agent. Rewards are differences between the values of an optimized metric between two timesteps, so that the optimized cumulative reward is equal to the final value of that metric. In all experiments we optimize for validation accuracy.

Actions are relative changes in each controlled hyperparameter. We predict the change independently for each hyperparameter, out of a discrete set of values $\{-50\%, -20\%, -5\%, 0, +5\%, +25\%, +100\%\}$. The action space conveys the intuition that a hyperparameter schedule should be approximately continuous in the log space, while allowing for both significant changes in the hyperparameters between timesteps, and for more fine-grained control. Opposite values of relative change cancel each other out, so that a random walk in the action space has a median relative change of approximately 1 for every hyperparameter.

3.1 Model-free approach

As a basic model-free approach to solving this problem, we use the Proximal Policy Optimization algorithm (Schulman et al., 2017). We set the discount factor γ to 1 to get an unbiased estimate of the expected return. This does not cause the return to diverge, as the rewards are bounded and the trajectory length is constant. To calculate advantages, we use Generalized Advantage Estimation (Schulman et al., 2015b) with $\lambda = 0.95$.

As a policy network, we use the decoder part of Transformer, without the embedding, so it accepts continuous input. The network has two heads, one predicting the distribution over actions, and the other the value (an estimate of the future return). We add to the PPO objective the L2 loss of the value head with weight 1 and an entropy bonus with weight 0.1.

3.2 Model-based approach

As a more sample-efficient, model-based approach, we use Simulated Policy Learning (SimPLe, Kaiser et al. (2017)). The algorithm consists of three phases, repeated in a loop, as shown in Figure 5. First, a set of trajectories is collected, either using the current policy network (which is randomly initialized) or using an external data-set. Then a predictive model of the environment is trained on the collected data. After that, the policy is trained using PPO in the environment simulated by the model. The improved policy is used to collect new data, and the loop continues.

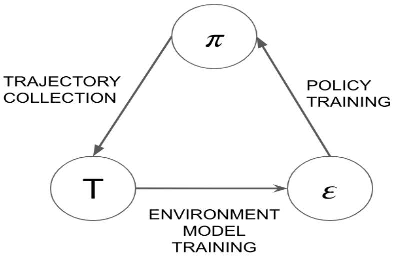

Figure 5: SimPLe algorithm. π denotes the policy, ϵ the environment model, and T the set of collected trajectories.

We use a simplified variant of SimPLe where we start the loop from training the model on a set of pre-collected data, and run only one iteration of the algorithm. Given a diverse dataset, this allows to learn a forecasting model accurate enough for policy training.

To be able to use the Transformer decoder as an environment model, we discretize the entire history of observations and actions and concatenate their representations into one long sequence $o_1 a_1 o_2 a_2 ... o_n$. Observations are discretized as described in subsection 2.1. Actions are already discrete, so we just rewrite them using one discrete symbol per each controlled hyperparameter. We train the model to predict just the observation symbols, masking out the loss terms corresponding to action symbols.

During inference, we calculate the reward based on the difference of an appropriate metric in two consecutive predicted observations.

4 Related Work

As described above, learning the training dynamics and utilizing it involves a mixture of time-series modeling, reinforcement learning (RL) and model-based RL. We discuss related works in this order.

Time-Series Modeling with Deep Learning. Time series forecasting has been traditionally studied using statistical models like ETS (Hyndman et al., 2008) and ARIMA (Box & Jenkins, 1994). There are a number of works that use feed forward neural networks (FFNNs) for time series forecasting, see a survey by Zhang et al. (1998). However, FFNNs break the input series into consecutive fixed size input windows, so the temporal order is ignored within each input window and every new input is considered in isolation. It is also not uncommon to use a hybrid approach of using FFNNs along with ARIMA or ETS as in Khashei & Bijari (2011) and Faruk (2010).

Recurrent Neural Networks (RNNs) are a more natural fit for sequence prediction tasks and have gained popularity for natural language processing tasks. They started to be applied to time-series later and were found to be competitive for point forecasts in a univariate context (Hewamalage et al., 2019). Recently Convolutional Neural Networks (CNNs) have also been used for time series forecasting, specially for capturing long-term dependencies, see van den Oord et al. (2016) and Lai et al. (2018). Other approaches have used Deep Belief Networks and Stacked Denoising Autoencoders to predict temperature and traffic flow (Romeu et al., 2013; Lv et al., 2015). Outside of language and generating audio, Transformers (Vaswani et al., 2017) have not been widely used for time-series prediction.

Reinforcement Learning DQN (Mnih et al., 2013) kicked off the field of Deep Reinforcement Learning by training a CNN on raw input pixels to predict the value of future rewards. Further work that built on top of DQN includes Double-DQN by van Hasselt et al. (2016), Dueling-DQN Wang et al. (2016) etc. On the other hand, pure policy optimization techniques like TRPO (Schulman et al., 2015a) and PPO (Schulman et al., 2017) directly represent the policy and optimize the policy parameters. PPO is quite simple to implement and works well on a variety of tasks: Atari, MuJoCo etc. A downside of pure policy gradient algorithms using on-policy learning is the large number of environment interactions needed to achieve satisfactory performance (sample complexity). Soft-Actor Critic (Haarnoja et al., 2018) is a recent development that uses ideas from both Q-learning and policy gradient methods to achieve better sample complexity.

Model-based Reinforcement Learning All RL methods mentioned above suffer from the requirement of a large number of interactions with the environment. This can be very costly, as in our case. The idea to use a model of the environment instead of the true one has been explored for a long time. For example, Holland et al. (2018) use a variant of Dyna Sutton (1991) to learn a model of the environment and generate experience for policy training in the context of Atari games. Outside of games, model-based reinforcement learning has been investigated at length for applications such as robotics Deisenroth et al. (2013). Though most of such works do not use image observations, several recent works have incorporated images into real-world robotic control (Finn et al., 2016; Finn & Levine, 2016; Babaeizadeh et al., 2017; Ebert et al., 2017; Piergiovanni et al., 2018; Paxton et al., 2018; Rybkin et al., 2018; Ebert et al., 2018) and simulated Watter et al. (2015); Hafner et al. (2018).

Hyperparameter Tuning Black-box hyperparameter tuning is extremely popular in industry and academia, examples include Google Vizier (Golovin et al., 2017), Hyperopt (Bergstra & Bengio, 2012), Spearmint (Snoek et al., 2012). These approaches assume the existence of a metric on a validation set. Sequential Model-Based Optimization (SMBO), (Hutter et al., 2011) is a family of methods that builds a model of the validation metric with hyperparameters as input, this model is usually trained on the previous trials of the hyperparameters. A large category of SMBO algorithms is Bayesian Optimization that builds a probabilistic model of the above function.A detailed survey about Bayesian Optimization can be found here Shahriari et al. (2016).

In Klein et al. (2017) the authors use Bayesian Neural Networks to predict the training curves of various models. In contrast to our work, they do not predict schedules of hyperparameters, so their models are not conditioned on these schedules.

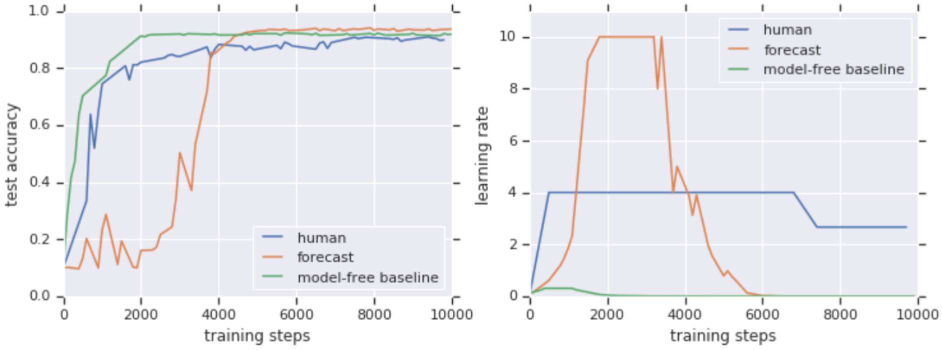

Figure 6: Test accuracy plot of Wide ResNet on CIFAR-10 and the corresponding learned learning rate schedule. The final accuracy is 93.6% for the forecaster, 91.8% for the model-free baseline and 90% for the human-defined schedule.

Population Based Training of Neural Networks Jaderberg et al. (2017) is another approach that jointly optimizes the parameters and hyperparameters of a network. It starts off with a population of agents, that after every few training iterations compare their fitness and mutate themselves using traditional evolutionary techniques (copying, mutation etc) and resume training from the next iteration onward, yielding an adaptive hyperparameter schedule that is similar to our work.

Also similar to our work, Xu et al. (2019) learn to tune hyperparameters during model training using reinforcement learning, but they only use a model-free approach and restrict themselves to controlling the learning rate.

5 Experimental Results

5.1 Wide ResNet on CIFAR-10

In our first experiment, we gather a data-set of 10K training runs of Wide ResNet on CIFAR-10 while varying the learning rate and weight decay used during training. The network we use is a Wide ResNet 4-28 and we train it for 10000 steps with batch size 512.

To gather the training curves, we run model-free PPO on 128 parallel environments. That means there are 128 concurrent workers training the network in each epoch, after which the policy for learning rate and weight decay tuning gets updated and another batch of 128 workers start training. We run 4 parallel runs of this, each for 20 epochs, resulting in 128*4*20 = 10240 training curves.

We then train on this set the forecasting model – a 3-layer Transformer with the loss described in Section 2.1. This forecaster fits a holdout set of curves quite well, as can be seen in Figure 1.

In order to get a quantitative verification of our forecaster, we train another PPO policy, this time using the forecaster as the environment. Since the forecaster is much faster, we train for 50 epochs. The resulting policy is even better than the one obtained using actual training curves, see Figure 6, which validates our claim that the forecaster is a useful model of deep learning dynamics.

Discussion While the above results attest to the quality of the forecaster, can we draw any insights from the learned policies? We analyzed the learning rate and weight decay schedules and there was a clear pattern employed by the forecaster. Namely: the schedule would increase learning rate not too rapidly at first, until accuracy gets reasonable (above 20-50%, stochastically). Then, it would increase the learning rate to the maximum, even at the cost of accuracy going slightly down and the loss increasing. It will recover in time though and only after a few hundred more steps the schedule would decrease the learning rate again to obtain the final result. This method, depicted on Figure 6, leads to higher validation accuracy than just keeping the accuracy rising. We conjecture that such schedule results in models that generalize better, but we leave it to future work to verify this conjecture.

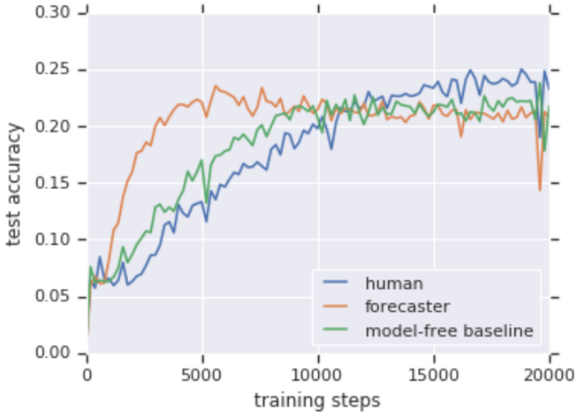

Figure 7: Test accuracy plot of the Transformer language model on Penn Treebank. The final accuracy is 21% for the forecaster, 23.8% for the model-free baseline and 23.2% for the human-defined schedule.

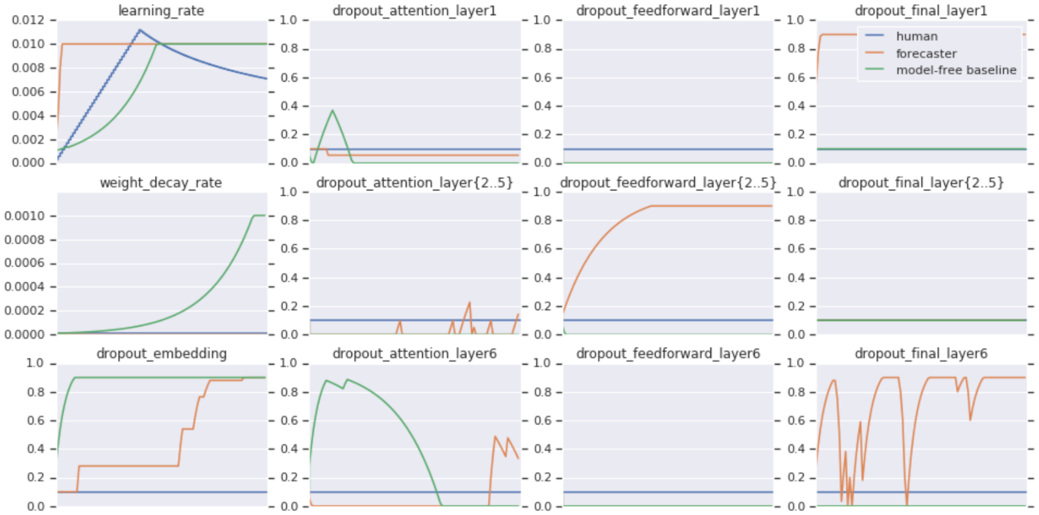

Figure 8: Hyperparameter schedules of the Transformer language model on Penn Treebank. See text for a discussion of the learned schedules.

5.2 Transformer on PTB

We repeat the experiment described above for Transformer models on the task of language modeling on the Penn Treebank corpus (PTB). In this model, in addition to learning rate and weight decay, we also tune 3 dropout parameters: the attention dropout, the dropout in the middle of feed-forward layers and the dropout at the final residual connection in a layer. Not only are there 3 different dropouts, we tune them separately for the first layer, last layer, and all other layers in the middle. Since we are training a Transformer model, these are layers 2-5.

The final accuracy curves for all models are depicted in Figure 7. Neither the model-free baseline nor the forecaster policy manages to out-perform the human baseline in this case. The forecaster overfits slightly and so its final score gets worse in the end, but it is close to the model-free baseline.

Discussion The schedules of these parameters used by the model-free baseline, the forecaster-trained model and human researchers are depicted in Figure 8. The forecaster clearly employs some interesting strategies. The dropout inside the feed-forward part of the middle layers is raised very heavily after the first few thousand steps, possibly because this 6-layer Transformer model has too many parameters for the PTB dataset. In the final layer, the feed-forward dropout is not changed but the final one oscillates and goes up in the end. Similarly for the dropout of the first layer of word embeddings: both the forecaster policy and the model-free one decide to raise it.

6 Conclusions

Modeling the learning dynamics of deep models is a complex problem, but a modification of the Transformer model with the discretization strategy we developed in Section 2.1 yields good results in many cases. The predicted curves not only have low L2 distance and look close to the real ones, as seen in Figure 1, but can also be used in a reinforcement learning setting to create parameter schedules.

Learning parameter scheduling policies using the forecaster is much more efficient than when running real experiments. A single inference of the forecaster, at batch 128, takes less than a minute even on a single GPU. Training the models on the other hand takes at least an hour and one needs 128 GPUs for that. So it is at least $60 \times 128 = 7680$ times more efficient to operate in simulation. We show experimentally that the forecasting model is indeed good enough to be used in this way.

One question that we leave for future work is how to effectively gather data for the forecaster model. Having a lot of training curves from varied models would allow pre-training the Transformer forecaster before using task-specific data. Pre-training with large data-sets has worked very well with Transformers, e.g., in the context of BERT and other NLP tasks, so we consider it a promising direction in this case too.

To pre-train the forecaster we will need to scale-up the effort of gathering data. Luckily, this can be a community effort as many people are training models and could benefit from better hyperparameter schedules. In preparation for this we have already released the code for our experiments as open-source[1] and we are gathering feedback as we extend its applicability.

Finally, aside from the applications to hyperparameter tuning, our forecaster can be applied to any time-series prediction task. We have not experimented with other tasks than the synthetic one, but our results in Section 2.1 are very encouraging and let us believe that Transformer-based models can achieve good results in many time-series prediction tasks.

References

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. *ICLR*, 2017.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012. URL `http://dl.acm.org/citation.cfm?id=2188395`.

George Edward Pelham Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 3rd edition, 1994. ISBN 0130607746.

Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2), 2013.

Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *CoRR*, abs/1710.05268, 2017.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Durdu Ömer Faruk. A hybrid neural network and ARIMA model for water quality time series prediction. *Eng. Appl. of AI*, 23(4):586–594, 2010. doi: 10.1016/j.engappai.2009.09.015. URL `https://doi.org/10.1016/j.engappai.2009.09.015`.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. *CoRR*, abs/1610.00696, 2016.

[1] Links to the repositories omitted here to preserve anonymity.

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, pp. 512–519, 2016. doi: 10.1109/ICRA.2016.7487173. URL `https://doi.org/10.1109/ICRA.2016.7487173`.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pp. 1487–1495. ACM, 2017. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3098043. URL `https://doi.org/10.1145/3097983.3098043`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1856–1865. PMLR, 2018. URL `http://proceedings.mlr.press/v80/haarnoja18b.html`.

Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *CoRR*, abs/1811.04551, 2018.

Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent neural networks for time series forecasting: Current status and future directions, 2019.

G. Zacharias Holland, Erik Talvitie, and Michael Bowling. The effect of planning shape on dyna-style planning in high-dimensional state spaces. *CoRR*, abs/1806.01825, 2018. URL `http://arxiv.org/abs/1806.01825`.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello (ed.), *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, volume 6683 of *Lecture Notes in Computer Science*, pp. 507–523. Springer, 2011. ISBN 978-3-642-25565-6. doi: 10.1007/978-3-642-25566-3_40. URL `https://doi.org/10.1007/978-3-642-25566-3_40`.

Robin Hyndman, Anne B. Koehler, J. Keith Ord, and Ralph D. Snyder. *Forecasting with Exponential Smoothing - The State Space Approach: Robin Hyndman*. Springer-Verlag Berlin Heidelberg, 2008. URL `https://www.springer.com/gp/book/9783540719168`.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017. URL `http://arxiv.org/abs/1711.09846`.

Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, and Konrad Czechowski Dumitru Erhan Chelsea Finn Piotr Kozakowski Sergey Levine Afroz Mohiuddin Ryan Sepassi George Tucker Henryk Michalewski Blazej Osinski, Roy H Campbell. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2017. URL `http://arxiv.org/abs/1903.00374`.

Mehdi Khashei and Mehdi Bijari. A novel hybridization of artificial neural networks and arima models for time series forecasting. *Appl. Soft Comput.*, 11(2):2664–2675, March 2011. ISSN 1568-4946. doi: 10.1016/j.asoc.2010.10.015. URL `http://dx.doi.org/10.1016/j.asoc.2010.10.015`.

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=S11KBYclx`.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research &#38; Development in Information Retrieval*, SIGIR '18, pp. 95–104, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5657-2. doi: 10.1145/3209978.3210006. URL `http://doi.acm.org/10.1145/3209978.3210006`.

Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic flow prediction with big data: A deep learning approach. *IEEE Trans. Intelligent Transportation Systems*, 16 (2):865–873, 2015. doi: 10.1109/TITS.2014.2345663. URL `https://doi.org/10.1109/TITS.2014.2345663`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL `http://arxiv.org/abs/1312.5602`.

Chris Paxton, Yotam Barnoy, Kapil D. Katyal, Raman Arora, and Gregory D. Hager. Visual robot task planning. *CoRR*, abs/1804.00062, 2018.

A. J. Piergiovanni, Alan Wu, and Michael S. Ryoo. Learning real-world robot policies by dreaming. *CoRR*, abs/1805.07813, 2018.

Pablo Romeu, Francisco Zamora-Martínez, Paloma Botella-Rocamora, and Juan Pardo. Time-series forecasting of indoor temperature using pre-trained deep neural networks. In Valeri Mladenov, Petia D. Koprinkova-Hristova, Günther Palm, Alessandro E. P. Villa, Bruno Appollini, and Nikola Kasabov (eds.), *Artificial Neural Networks and Machine Learning - ICANN 2013 - 23rd International Conference on Artificial Neural Networks, Sofia, Bulgaria, September 10-13, 2013. Proceedings*, volume 8131 of *Lecture Notes in Computer Science*, pp. 451–458. Springer, 2013. ISBN 978-3-642-40727-7. doi: 10.1007/978-3-642-40728-4_57. URL `https://doi.org/10.1007/978-3-642-40728-4_57`.

Oleh Rybkin, Karl Pertsch, Andrew Jaegle, Konstantinos G. Derpanis, and Kostas Daniilidis. Unsupervised learning of sensorimotor affordances by stochastic future prediction. *CoRR*, abs/1806.09655, 2018.

John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, pp. 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2016. doi: 10.1109/JPROC.2015.2494218. URL `https://doi.org/10.1109/JPROC.2015.2494218`.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pp. 2960–2968, 2012. URL `http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms`.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL `http://arxiv.org/abs/1609.03499`.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Dale Schuurmans and Michael P. Wellman (eds.), *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pp. 2094–2100. AAAI Press, 2016. ISBN 978-1-57735-760-5. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need`.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In Maria-Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 1995–2003. JMLR.org, 2016. URL `http://proceedings.mlr.press/v48/wangf16.html`.

Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin A. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 2746–2754, 2015.

Zhen Xu, Andrew M. Dai, Jonas Kemp, and Luke Metz. Learning an adaptive learning rate schedule, 2019.

Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, 14(1):35 – 62, 1998. ISSN 0169-2070. doi: https://doi.org/10.1016/S0169-2070(97)00044-7. URL `http://www.sciencedirect.com/science/article/pii/S0169207097000447`.