Predictive Skill of Deep Learning Models Trained on Limited Sequence Data

Kookjin Lee, Jaideep Ray, and Cosmin Safta
ABSTRACT

In this report we investigate the utility of one-dimensional convolutional neural network (CNN) models in epidemiological forecasting. Deep learning models, especially variants of recurrent neural networks (RNNs) have been studied for influenza forecasting, and have achieved higher forecasting skill compared to conventional models such as ARIMA models. In this study, we adapt two neural networks that employ one-dimensional temporal convolutional layers as a primary building block – temporal convolutional networks and simple neural attentive meta-learner – for epidemiological forecasting and test them with influenza data from the US collected over 2010-2019. We find that epidemiological forecasting with CNNs is feasible, and their forecasting skill is comparable to, and at times, superior to, RNNs. Thus CNNs and RNNs bring the power of nonlinear transformations to purely data-driven epidemiological models, a capability that heretofore has been limited to more elaborate mechanistic/compartamental disease models.
ACKNOWLEDGMENT

This report describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525.
CONTENTS

1. Introduction .................................................. 9

2. Methods and materials ..................................... 11
   2.1. Recurrent Neural Networks (RNN) .................... 11
       2.1.1. Long Short-Term Memory (LSTM) ................ 12
       2.1.2. Gated Recurrent Unit (GRU) ..................... 13
   2.2. Sequence-to-Sequence (Seq2Seq) Model .............. 13
   2.3. Convolutional Networks ................................ 14
       2.3.1. Temporal Convolutional Networks (TCNs) ....... 14
       2.3.2. Simple Neural Attentive Meta-Learner (SNAIL) .. 14
   2.4. Materials .............................................. 15
       2.4.1. Preprocessing ...................................... 16

3. Results and discussion ..................................... 17
   3.1. Training neural networks ............................... 17
   3.2. Performance metrics ................................... 17
   3.3. One-week-ahead predictions results .................. 18
       3.3.1. LSTM and GRU .................................. 18
       3.3.2. Seq2Seq ......................................... 20
       3.3.3. TCN ............................................. 21
       3.3.4. SNAIL ........................................... 23
       3.3.5. Performance comparisons between Seq2Seq, TCN, and SNAIL ........................................ 23
   3.4. N-weeks-ahead predictions ............................. 26
   3.5. Discussion .............................................. 28

4. Conclusion .................................................. 33

References ..................................................... 34
LIST OF FIGURES

Fig 2-1. An example RNN architecture with two hidden layers: a folded representation (left) and an unfolded representation (right). ................................. 12
Fig 2-2. An example Seq2Seq architecture with one hidden layer in the encoder and the decoder. The context vector is depicted in the middle. ......................... 13
Fig 2-3. A temporal convolutional network with layers corresponding to exponentially increasing dilation factors $d = 1, 2, 4$. .................................................. 14
Fig 2-4. A schematic view of SNAIL with one TCBLOCK and two ATTENTION BLOCKS. . 15
Fig 2-5. An example illustration for applying a sliding window to a %ILI curve corresponding to California, from the 2013-2014 season to the 2014-2015 season. 16

Fig 3-1. Four performance indicators for one-week-ahead predictions obtained with RNN-LSTM and RNN-GRU. Smaller values are preferable for RMSE, MAPE, L2E, and larger values are better for PCORR. ................................................................. 19
Fig 3-2. Four performance metrics for one-week-ahead predictions obtained by using RNN-LSTM and RNN-GRU for varying number of layers $n_h = \{2, 3, 4, 5\}$. ............... 20
Fig 3-3. Four performance indicators for one week-ahead predictions obtained with RNN-GRU and the Seq2Seq model for varying number of layers $n_h = \{2, 3, 4, 5\}$. Only the best performing values of $w_{\text{hist}}$ are presented. ......................................................... 21
Fig 3-4. Four performance indicators for one week-ahead predictions obtained with TCNs with varying numbers of residual blocks $n_R = \{5, 6, 7, 8\}$ and kernel size $k = 4$. 22
Fig 3-5. Four performance metrics for one-week-ahead predictions obtained by using the SNAILs for varying number of TC layers $n_{\text{TC}} = \{6, 7, 8\}$ with the kernel size $n_{\text{key}} = \{16, 32\}$. ................................................................. 24
Fig 3-6. The best performances achieved by Seq2Seq, TCN, and SNAIL. Four performance indicators, RMSE, MAPE, L2E, and PCORR, are measured for each state. On the horizontal axis, the states are in a decreasing order w.r.t. the relative performance of SNAIL against Seq2Seq. ..................................................... 25
Fig 3-7. The boxplots of all four performance metrics measured by using Seq2Seq, TCN, and SNAIL. The boxplots summarize statistics of the data shown in Fig 3-6. The middle black and yellow lines indicate medians and means, respectively, and the boxes indicates the interquartile ranges. Top half: the entire range of the boxplots. Bottom half: the IQR of the boxplots excerpted. ........................................ 27
Fig 3-8. The proportion of the states where the former network outperforms the later network in 0, 1, 2, 3, or all 4 error indicators (RMSE, MAPE, L2E, and PCORR). The numbers are shown in this figure are collected from the results shown in Fig. 3-6. 28
Fig 3-9. Example plots of %ILI and predictions made by Seq2Seq and SNAIL for two seasons 2017-2018 (validation) and 2018-2019 (test). The original %ILI is depicted in solid black lines and the prediction results of Seq2Seq and SNAIL are depicted in solid red lines and dashed green lines.

Fig 3-10. Example plots of %ILI and predictions made by Seq2Seq and SNAIL for two seasons 2017-2018 (validation) and 2018-2019 (test). The original %ILI is depicted in solid black lines and the prediction results of Seq2Seq and SNAIL are depicted in solid red lines and dashed green lines.

Fig 3-11. Example plots of %ILI and predictions made by Seq2Seq and SNAIL for two seasons 2017-2018 (validation) and 2018-2019 (test). The original %ILI is depicted in solid black lines and the prediction results of Seq2Seq and SNAIL are depicted in solid red lines and dashed green lines.

Fig 4-1. LSTM cell diagram.

Fig 4-2. GRU cell diagram.

Fig 4-3. TCN residual block. The residual block consists of two layers of dilated convolutional layers, each of which followed by weight normalization, ReLU, and Dropout. Then the resulting quantity of the dilated convolutions and the input are summed element-wise. If the input and the output sizes do not match, an optional $1 \times 1$ convolution can be used.

Fig 4-4. SNAIL DENSE BLOCK. DENSE BLOCK consists of two parallel dilated causal convolutions. The outputs of the convolutional layers are multiplied element-wise and then the resulting quantity is concatenated with the input.

Fig 4-5. SNAIL DENSE BLOCK. DENSE BLOCK consists of two parallel dilated causal convolutions. The outputs of the convolutional layers are multiplied element-wise and then the resulting quantity is concatenated with the input.
Table 3-1. Neural network configurations: the number of GRU layers $n_k$, the number of
RESBLocks $n_R$, the number of TCBLocks $n_{TC}$, the kernel size $k$, the number
of kernel filters $n_k$, and the size of key $n_{key}$ in SNAIL. For TCN and SNAIL, the
value of $w_{hist}$ is omitted as setting $w_{hist} = 128$ yields the best results. For TCN,
$n_k = 4$. ................................................................. 23

Table 3-2. The best performances achieved by Seq2Seq, TCN, and SNAIL. The perfor-
mance metrics are averaged over 49 states. The symbols ↓ and ↑ indicate that the
lower and higher values correspond to the better performance. ................. 26

Table 3-3. The performance metrics measured by Seq2Seq, TCN, and SNAIL for varying
prediction horizons \{1, 2, 3, 4\}. The same sets of network hyperparameters for
each model are used for varying horizons. The downward/upward arrows in
parentheses indicate smaller/larger values are preferred. ....................... 31
1. INTRODUCTION

Seasonal influenza results in 9–45 million illnesses, between 140k–180k hospitalizations, and between 12k–61k deaths annually since 2010. It is the subject of intensive and continuous surveillance by the US Center for Disease Control [34]. Forecasting influenza outbreaks is important as it allows us to plan for medical resource demands [16]. To address this problem, CDC has been making efforts on Flu Forecasting [9], of which the goal is to predict the disease dynamics, e.g., predicting the timing, peak, and intensity of flu season, so that its impact can be reduced. Since a competition, “Predict the Influenza Season Challenge” (see [10]), hosted by CDC in 2013, CDC has encouraged researchers to develop models to make accurate predictions of influenza activities for a number of weeks ahead. Along with traditional mechanistic modeling approaches, there have been many studies on developing statistical forecast models using historical flu activity data: auto-regressive modeling [8], Google Flu Trend [17], Google search data [54, 53], and refined models using structural spatiotemporal synchronicities [15, 32], to name a few.

More recently, with the advancements in deep learning and the availability of high-spatial-resolution data, i.e., ten US HHS-regions-level data (regions defined by the United States Department of Health & Human Services) and state-level data for all US states, there are many studies on applying deep learning techniques for flu forecasting. Among several artificial neural network architectures, for handling time series, there are largely two types of architectures that have been extensively studied: recurrent neural networks (RNNs), which consist of shared and recurring units, and convolutional neural networks (CNNs), which consist of layers of convolutional operations. RNNs are defined as a family of neural networks for processing “sequential data” [21] and have shown a significant impact in applications such as language modeling [42, 22], machine translation [3], and polyphonic music modeling [13]. RNNs and their variants also have been actively explored in flu (%ILI) prediction studies [47, 56, 52, 46, 49, 31, 51].

CNNs are often considered to be a specialized neural networks for processing images, exploiting the spatial local correlations. CNNs, however, also have been applied to sequence modeling in many applications including speech recognition [48] and natural language processing [14, 26, 28, 57, 25]. More recently, variants of CNNs that are specially designed for sequence modeling using operations, “temporal” convolutions, have demonstrated an improved performance in audio synthesis [36], machine translations [19, 26], and various other sequence modeling [4] compared to earlier NN models. In particular, the extensive experiments in [4] demonstrated that relative simple temporal convolution architectures could exhibit substantially longer memory than RNNs. Despite these successes, temporal convolutions have rarely been explored in modeling infectious disease models.
In this study, we model flu forecasting as a sequence modeling task, explore different NN architectures, essentially variants of RNNs and CNNs, and then compare performances of all models with extensive experiments. Following [17, 54, 32], we utilize the CDC data on the percentage of the number of influenza-like illness (ILI) patients over the total number of outpatient visits, which is denoted by %ILI; this information can be used as a proxy of the flu activity in the population and help hospital officials allocate appropriate resources in preparation for increased patient visits to hospital facilities. This study considers only the state-level %ILI from CDC without any external data (e.g., external factors that might affect the flu activity or additional internet data that can be utilized to forecast).

Both statistical time-series models [18] e.g., ARIMA, and mechanistic/compartmental models [27] are used in forecasting epidemics. Compartmental models require one to model population mixing and disease phenomenology (e.g., incubation, prodrome, existence and role of asymptomatic individuals etc.) which can be difficult in case of novel diseases. However, modern information technology allows the (very) quick and comprehensive collection of basic epidemiological data e.g., detected or diagnosed cases, at the country-, regional- and often, city-scale, as evidenced during the ongoing COVID-19 pandemic [2, 1], and purely data-driven methods could be used for forecasting and practical resource planning scenarios even without fully understanding the disease dynamics. This raises the question whether more complex data-driven models, such as CNNs and RNNs, can be trained on the big data to provide better forecasts than linear ARIMA models. RNNs, in fact, have been used for forecasting COVID-19 [30], and comparisons of RNN-based forecasting for influenza have shown that neural networks perform far better than ARIMA models [51]. In this paper we investigate whether CNNs, one-dimensional temporal convolutions, in particular, could be used for the same purpose.

An outline of this report is as follows. In Chapter 2 we review neural networks models with and without temporal convolutions. Next, we describe datasets in detail and training strategies for neural networks in Section 3.1. We present results of experiments of one-week ahead predictions and multi-week ahead predictions in Sections 3.3 and 3.4. We conclude with a discussion in Section 3.5.
2. METHODS AND MATERIALS

Assume that we have a sequence of input data \( \{x_1, \ldots, x_t\} \) with the time index \( t \), which can be obtained for example from measurements, and we wish to predict future sequence, \( \{x_{t+1}, x_{t+2}, \ldots\} \) via a sequence modeling. In a general form, sequence modeling consists of learning a function \( f(\cdot; \Theta) \) such that

\[
\{o_1, \ldots, o_{n_{\text{out}}} \} = f(x_1, \ldots, x_{n_{\text{in}}}; \Theta),
\]

where \( \Theta \) is a set of parameters to be learned, and \( \{o_1, \ldots, o_{n_{\text{out}}} \} \) is a sequence of outputs. Here, \( n_{\text{in}} \) and \( n_{\text{out}} \) are not necessarily to be the same. In a supervised learning setting, there are target (or reference) variables \( \{y_1, \ldots, y_{n_{\text{out}}} \} \), on which the model \( f(\cdot; \Theta) \) can be trained to produce outputs \( \{o_1, \ldots, o_{n_{\text{out}}} \} \) to match target variables \( \{y_1, \ldots, y_{n_{\text{out}}} \} \). In a forecasting scenario, where we attempt to predict future data, the target variables can be set as, for example, \( y_1 = x_{t+1}, y_2 = x_{t+2} \) for 2-weeks ahead prediction. Given input data and target variables, an optimal set of parameters can be learned by minimizing a certain loss function \( L(y_1, \ldots, y_{n_{\text{out}}}, o_1, \ldots, o_{n_{\text{out}}}) \), which measures the discrepancy between the output \( \{o_1, \ldots, o_{n_{\text{out}}} \} \) and the target \( \{y_1, \ldots, y_{n_{\text{out}}} \} \) in a certain measure.

The parameterized function \( f(\cdot; \Theta) \) can take various forms such as ones based on statistical models or deep-learning-based models. In this study, we are particularly interested in deep-learning-based models such as recurrent neural networks (RNNs), sequence-to-sequence (or encoder-decoder type) RNN variants, and neural networks based on temporal-causal convolutions. Our goal in this study is to perform a comprehensive empirical evaluation on performances of various types of neural networks for near-term forecast of influenza-like illness. In the following, we briefly review each of considered deep-learning-based models.

2.1. Recurrent Neural Networks (RNN)

A recurrent neural network (RNN) \cite{40} is a type of neural networks specially modeled to process a sequence of values and, thus, has a good fit for predicting time-sequences. At time index \( \tau \), RNNs typically operate on a sequence of data \( x_\tau \), update hidden states \( h_\tau^{(1)}, h_\tau^{(2)}, \ldots \), and produce an output \( o_\tau \) by applying the same function parameterized by a set of neural network weights:

\[
o_\tau, h_\tau^{(1)}, h_\tau^{(2)}, \ldots = \text{RNN}(x_\tau, h_{\tau-1}^{(1)}, h_{\tau-1}^{(2)}, \ldots; \Theta), \tag{2.1.1}
\]

where \( h^{(\ell)} \) denotes the \( \ell \)-th hidden layer. Fig 2-1 depicts an example RNN architecture with two hidden layers and Eq. (2.1.1) describes a folded representation of RNN, shown in the left frame of Fig 2-1.
Hidden layers and outputs of RNNs are computed in sequence. Assume an RNN architecture with one hidden layer. Then, starting from a given initial hidden state $h_0^{(1)}$, the forward pass of the RNN computes subsequent hidden layers by applying an affine transformation to a current input and the previous hidden states followed by a nonlinear activation, e.g.,

$$h_\tau^{(1)} = \sigma(W_{xh}x_\tau + W_{hh}h_{\tau-1}^{(1)} + b),$$

where $W_{xh}$, $W_{hh}$, and $b$ are the learnable parameters and $\sigma$ is a nonlinear activation function.

Standard RNNs typically suffer from vanishing/exploding gradient problems [5, 24, 37]. To cope with this difficulty, architectures using gating mechanisms including long short-term memory (LSTM) [24] and gated recurrent unit (GRU) [12] have been proposed and extensively used in several applications. The core idea is to create paths whose gradients do not vanish or explode as RNN cells become far apart.

### 2.1.1. Long Short-Term Memory (LSTM)

In LSTM recurrent networks, the above issue is resolved by adding LSTM cells, which have an internal recurrence to update cell states, generating paths where the gradient can flow over longer durations [24], as opposed to standard RNNs that only have recurrence over the hidden states. In addition, by introducing gating mechanisms which typically consists of input, output, and forget gates, the flow of information can be effectively controlled [24, 20]. As the name indicates, the input and the output gates control the amount of information that gets transferred from input to output, respectively, and the forget gate controls the extent to which internal cell state remains unchanged. When the sigmoid function is used, each gate produces a value between $[0, 1]$, where 0 and 1 are two extreme values indicating that no or all information will be passed, respectively.
2.1.2. **Gated Recurrent Unit (GRU)**

GRU is another gating mechanism without internal cell states and with a smaller number of gates; GRU consists of reset and update gates \[12\]. The reset gate controls information flow, choosing which parts of the previous state to pass to generate a new target state. Then the update gate controls the extent to which the previous state remains and the new target state replaces the previous state to generate a new state.

Compared to LSTM, GRU has a simpler architecture: it does not have a cell state and it operates only on two gates (as opposed to three gates in LSTM). In general, LSTM is known to be strictly more expressive and it has been demonstrated that LSTM outperforms GRU on many different applications \[7, 50\]. In certain tasks, however, GRU performs similar to LSTM \[39\], or performs even better on small datasets \[13\]. We refer readers to the Appendix for details on LSTM, GRU, and their comparisons.

2.2. **Sequence-to-Sequence (Seq2Seq) Model**

The next model we consider is a sequence-to-sequence (Seq2Seq) model (or, often called encoder-decoder model): Seq2Seq architecture is designed to train a mapping from an input sequence to an output sequence, where the input and the output sequences do not necessarily have the same length \[12, 43\]. For handling variable-length sequences, the Seq2Seq architecture consists of an encoder and a decoder, which are typically RNN-type architectures (with LSTM cells or GRUs). Because the input and the output of ILI forecasting can be two variable-length sequences, the Seq2Seq architecture is a naturally good fit. The encoder processes an input sequence to produce a single vector called a context vector, which is typically a function of the last hidden state of the encoder. Then the decoder generates an output sequence conditioned on the single context vector. Fig. 2-2 depicts an example Seq2Seq architecture. We leave more details to the Appendix.

![Fig 2-2. An example Seq2Seq architecture with one hidden layer in the encoder and the decoder. The context vector is depicted in the middle.](image-url)
2.3. Convolutional Networks

As a next set of sequence modeling approaches, we present two neural networks: temporal convolutional networks and simple neural attentive meta-learners, of which the main component is a temporal convolution (TC) layer. The TC layer refers to an one-dimensional causal convolution layer; an input to this layer is 1D time-sequenced data, which may consists of multi channels, same as in an input to a regular 1D convolutional layers, and as the term, “causal”, indicates, an output of the TC layer at time $t$, $o_t$, is produced by applying convolution kernel filters to the input data at time $t$ and earlier times, i.e. $\{\ldots, x_{t-1}, x_t\}$. To utilize a longer history from the input data without making the depth of neural network too deep, a sequence of dilated convolutions with increasing dilation factors can be used as in Refs. 36, 55. In a typical setting, the dilation factor starts with $d = 1$ and increases exponentially, i.e., $d = O(2^\ell)$ at level $\ell$ of the network. Thus, the receptive field of the network can be controlled by the kernel size $k$ and the dilation factor $d$. As for RNNs, TC layers can handle arbitrary-length input sequence. Fig 2-3 illustrates an example of TC layers with several dilation factors.

![Fig 2-3. A temporal convolutional network with layers corresponding to exponentially increasing dilation factors $d = 1, 2, 4$.](image)

2.3.1. Temporal Convolutional Networks (TCNs)

Among several neural network models based on the temporal convolutions, we explore the temporal convolution networks (TCNs) in this study since TCNs provide a relatively simple, but flexible architecture. In addition to the convolution layers, TCNs typically employ residual blocks [23], which we denote by RESBLOCK, to stabilize the computation. Each RESBLOCK consists of (1) two layers of dilated causal convolution, where each layer is followed by weight normalization, ReLU [35], and dropout, and (2) the identity mapping from the input to the block (optionally, an $1 \times 1$ convolutional layers can be employed to match the input and the output shapes so that the element-wise summation can be performed). We refer readers to Appendix for details of RESBLOCK.

2.3.2. Simple Neural Attentive Meta-Learner (SNAIL)

As described above, TCNs handle long sequences by employing exponentially increasing dilation factors. This may lead to coarser access to inputs and, consequently, bounded network capacity.
To resolve this issue, a simple neural attentive meta-learner (SNAIL) [33] proposed to combine temporal (causal) convolutions with a soft attention mechanism that is similar to Ref. [45]. SNAIL interleaves TC layers with attention layers so that the model can learn to (1) extract features from the current and previous input features/data via the TC layers and (2) identify more important input features/data over other elements in long sequences via the attention layers.

These two main ingredients of SNAIL can be summarized by two separate functions, TCBLOCK and ATTENTIONBLOCK. In TCBLOCK, a series of TC layers with exponentially increasing dilation factors is used; at each TC layer, an output is computed using the gated activation function [36, 44] (as opposed to the simple TCNs) and then is concatenated with an input. In ATTENTIONBLOCK, a causal attention mechanism is employed to point out the data points in the input sequence that should be more emphasized. The term “causal” indicates that the attention mechanism only looks at the current time input data and its previous time input data. Fig 2-4 depicts a schematic view of SNAIL with one TCBLOCK and two ATTENTIONBLOCKS. We refer readers to the Appendix for details of TCBLOCK and ATTENTIONBLOCK.

**Fig 2-4. A schematic view of SNAIL with one TCBLOCK and two ATTENTIONBLOCKS.**

2.4. Materials

The Influenza Division at the Center for Disease Control (CDC) reports a weekly U.S. influenza surveillance data (FluView); in particular, we are interested in information on outpatient visits to health care providers for influenza-like illness (ILI), which is collected through the U.S. Outpatient Influenza-like Illness Surveillance Network (ILINet). ILINet consists of outpatient healthcare provider in all 50 states and records the total number of patient visits and the number of those patients with ILI symptoms, which are defined as combinations of fever (temperature over 100°F), a cough, and/or a sore throat without a known cause other than influenza.

Our particular interest lies on the percentage of the number of ILI patients over the total number of outpatient visits, which is denoted by %ILI (% unweighted ILI). This information is available per state each week and can be downloaded from the application, FluView Interactive [11].
2.4.1. Preprocessing

We downloaded state-level %ILI from week 40 of 2010 to week 39 of 2019 (9 seasons), and preprocessed to keep %ILI only during the influenza season (i.e., week 40 of a given year to week 20 of a subsequent year). Consequently, each influenza season consists of 33 weeks, except for 2014-2015 season, which has 34 weeks due to the 53rd week in 2014. Among the 50 states, Florida is excluded from the study because there is no data reported. We rescale the data to have values in range [0,1] using min-max scaling. Note that predictions made by each neural network are then scaled back to the original range by applying the inverse of the min-max scaling operator in a post-processing step and the resulting quantities are used to compute performance metrics, which will be introduced in the next section. Moreover, we generate a binary mask, which has the same length as the %ILI data, to indicate missing, unreported, or a weekly report with zero values.

Following the preprocessing step shown in Ref. [51], we concatenate 9 seasons of %ILI per each state and use a fixed-length sliding window to generate subsequences for supervised-learning settings (see Fig. 2-5). The window consists of two subwindows of length \( w_{\text{hist}} \) and \( w_{\text{pred}} \) (in weeks), respectively, and sequence models are expected to make predictions on \( w_{\text{pred}} \) weeks of %ILI given \( w_{\text{hist}} \) weeks of historical observations on %ILI.

![Fig 2-5. An example illustration for applying a sliding window to a %ILI curve corresponding to California, from the 2013-2014 season to the 2014-2015 season.](Image)

We then split the data into training/validation/test sets. The training set, the validation set, and the test set consist of subsequences obtained from 2010 to 2017 (7 seasons), from the 2017-2018 season, from the 2018-2019 season, respectively. For \( w_{\text{pred}} \)-weeks-ahead predictions, the validation set and the testing set are designed to include subsequences of which the last data point belongs to the 2017-2018 season and the 2018-2019 season, respectively.
3. RESULTS AND DISCUSSION

We now assess the performance for %ILI forecasting for the neural networks described in the previous section. We construct all neural network architectures using PyTorch 1.12 [38]. Before presenting the results, we describe neural network training strategies employed in this work and performance metrics used to evaluate the trained neural networks on the test dataset.

3.1. Training neural networks

For training neural networks, we consider the Gaussian negative log-likelihood as the loss function, and attempt to minimize it via a gradient-based optimization method [6]. In particular, we employ a variant of stochastic gradient descent method called Adamax [29] with an initial learning rate of $10^{-2}$ and a mini-batch size to 50. The maximum number of epochs is set to 50. At each epoch, the validation loss is computed on the validation set, and the best performing network weights on the validation set are chosen to try the model on the test data. Moreover, we use an early-stopping strategy; the training stops if there is a certain number of epochs where the optimizer fails to find improvements in validation loss. From our empirical findings, in general, the early-stopping strategy does not help to generalize the performance of the RNN variants within the pre-specified maximum number of epochs, whereas this approach helps to generalize the performance of the TC variants. Thus, we use the early-stopping strategy only for TCNs and SNAILs, and set the number of consecutive epochs to 3 for the early-stopping criterion.

We also employ a grid search for optimal hyperparameter values. Although there are more advanced hyperparameter search algorithms such as Bayesian hyperparameter optimization [41], here we use grid search to put more emphasis on the impact of these hyperparameters on the performance of each models and attempt to draw interpretations on the impact of these changes on model improvements.

3.2. Performance metrics

We consider four different metrics to measure the discrepancy between the predicted sequence and the target sequence and assess the performance of each neural network models

- root mean-square error (RMSE)

$$\left( \frac{1}{N} \sum_{i=1}^{N} (x_i - \tilde{x}_i)^2 \right)^{1/2}$$
• mean absolute percentage error (MAPE)
\[
\frac{1}{N} \sum_{i=1}^{N} \frac{|x_i - \tilde{x}_i|}{x_i + 1} \times 100
\]

• relative error measured in the Euclidean norm (L2E)
\[
\frac{\left(\sum_{i=1}^{N} (x_i - \tilde{x}_i)^2\right)^{1/2}}{\left(\sum_{i=1}^{N} x_i^2\right)^{1/2}}
\]

• Pearson correlation coefficient (PCORR)
\[
\frac{\sum_{i=1}^{N} (x_i - m_x) (\tilde{x}_i - m_{\tilde{x}})}{\sqrt{\sum_{i=1}^{N} (x_i - m_x)^2 \sum_{i=1}^{N} (\tilde{x}_i - m_{\tilde{x}})^2}}
\]

RMSE and L2E computes the averaged and the relative sum of squared errors, respectively. MAPE computes the errors in L1-norm and then report the error in a percentage. Following [49], the denominator is smoothed by adding 1 to avoid zero values. Lastly, PCORR measures how correlated two sequences are and the resulting quantity lie between [-1,1]. The closer PCORR is to 1, the more correlated two sequences are. Note that L2E and MAPE are relative measures, whereas RMSE is not: RMSE measures the squared differences of \(x_i\)’s where \(x_i < 16\) in test sets.

3.3. One-week-ahead predictions results

For this experiment, we use varying lengths of historical observations, \(w_{\text{hist}} = \{16, 32, 64, 128\}\), to make one-week-ahead predictions, i.e., \(w_{\text{pred}} = 1\). We will explore performances of models discussed above with the error indicators presented in the previous section. Unless otherwise specified, we compute errors for 49 states separately and then average over 49 states. We begin by comparing RNN-LSTM and RNN-GRU models.

3.3.1. LSTM and GRU

For both RNN-LSTM and RNN-GRU, we consider the same supervised learning setting. As described in the preprocessing section, we consider the subsequences of length \(w_{\text{hist}} + w_{\text{pred}}\). We denote the \(i\)-th input data sequence by \(x_1^{(i)}, \ldots, x_{w_{\text{hist}}}^{(i)}\), where \(x_{\tau}^{(i)}\) is used as the input to the \(\tau\)-th step of the RNNs. At the \(\tau\)-th RNN step, the network produces output \(o_{\tau}^{(i)}\), which attempts to match \(y_{\tau}^{(i)} = x_{\tau+1}^{(i)}\) so that, in the last RNN step, the quantity that the network tries to predict is produced. We train both RNN-LSTM and RNN-GRU by minimizing the negative log-likelihood function. In the first experiment, we consider RNN-LSTM and RNN-GRU with two hidden layers. For both RNNs, the first layer consists of 32-dimensional hidden units (and a 32-dimensional cell
state in LSTM) and the second layer consists of 16-dimensional hidden units (and a 16-dimensional cell state in LSTM). Then the output of the last layer is connected to a dense network with one hidden layer, which consists of 100 units, followed by a ReLU activation. We have tested these two RNNs on the datasets obtained by varying the lengths of historical observations $w_{\text{hist}} = \{16, 32, 64, 128\}$. Fig 3-1 reports the four performance indicators measured on the testing set and, in most cases, RNN-GRU results are better.

Next we consider RNNs with varying number of LSTM layers and GRU layers as $n_h = \{2, 3, 4, 5\}$. We fixed the size of hidden units in the last layer to be 16 and those of other layers to be 32. We have tested the new architectures with varying lengths of historical observations $w_{\text{hist}} = \{16, 32, 64, 128\}$ as described above. From these experiments, we observe that RNN-LSTM performs best with $w_{\text{hist}} = 64$ and RNN-GRU performs best with $w_{\text{hist}} = 32$. Fig. 3-2 reports the performance metrics measured for the LSTM and the GRU with their corresponding best settings. For both RNNs, we can observe that all four indicators improve, except for MAPE of LSTM, as $n_h$ increases. Although other results are not reported, we also have observed that increasing $w_{\text{hist}}$ to 128 significantly decreases the performance of both LSTM and GRU suggesting that now the models are overfit. Overall, RNN-GRU outperforms RNN-LSTM in all error metrics and, thus, we consider only GRU for the remainder of this report.
Fig 3.2. Four performance metrics for one-week-ahead predictions obtained by using RNN-LSTM and RNN-GRU for varying number of layers $n_h = \{2, 3, 4, 5\}$.

3.3.2. Seq2Seq

Seq2Seq-type networks consists of an encoder and a decoder and both of them are modeled as RNN-GRU in this study. As for the previous settings used in the comparisons of RNN-LSTM and RNN-GRU, the $i$-th input sequence is denoted by $x_1^{(i)}, \ldots, x_{w_{hist}}^{(i)}$ and the encoder RNN network operates on this input sequence to produce the context vector. The decoder receives the last element in the input sequence (i.e., $x_{w_{hist}}^{(i)}$) as its input as well as the context vector to produce the output $o_1^{(i)}$, which is used to attempt $y_1^{(i)} = x_{w_{hist}+1}^{(i)}$.

The encoder and the decoder are designed to have a symmetric architecture in terms of the number of hidden layers. For instance, if the encoder consists of two hidden layers of unit sizes 32 and 16, respectively, the decoder consists of two hidden layers of unit sizes 16 and 32. As in the previous experiments with LSTM and GRU, we have tested the Seq2Seq architecture with a varying number of GRU layers by setting $n_h = \{2, 3, 4, 5\}$; the last hidden layer of the encoder and the first hidden layer of the decoder are of dimension 16, and the other hidden layers are of dimension 32. We again consider varying lengths of historical observations $w_{hist} = \{16, 32, 64, 128\}$.

In Fig. 3-3 we present results obtained with the Seq2Seq architectures with the two best performing values of $w_{hist} = \{64, 128\}$, and compare those results with the best results with
RNN-GRU models ($w_{\text{hist}} = 32$) presented in Fig. 3-2. The experiments show that Seq2Seq performs better with longer sequences (i.e., $w_{\text{hist}} = \{64, 128\}$) and, as shown in the experiments with RNN-GRU, performs better with the larger networks (i.e., $n_h \geq 3$), in general; we note that there is a single exception $n_h = 4$ and $w_{\text{hist}} = 128$. By comparing the Seq2Seq results with GRU results in Fig. 3-2, we observe that the Seq2Seq architectures outperform the standard RNN-GRU in several cases, and that the Seq2Seq architectures with $n_h = 3$ and $w_{\text{hist}} = 64$ produce the best results.

![Graphs showing RMSE, MAPE, L2E, and PCORR for different values of $n_h$ and $w_{\text{hist}}$.](image)

Fig 3-3. Four performance indicators for one week-ahead predictions obtained with RNN-GRU and the Seq2Seq model for varying number of layers $n_h = \{2, 3, 4, 5\}$. Only the best performing values of $w_{\text{hist}}$ are presented.

### 3.3.3. TCN

Next we discuss results obtained with TCN models. In this experiment, we consider the same supervised learning settings for the input and the target data, i.e., the $i$-th input data sequence is $\{x^{(i)}_\tau\}_{\tau=1}^{w_{\text{hist}}}$ and the $i$-th target data sequence is $\{y^{(i)}_\tau\}_{\tau=1}^{w_{\text{hist}}}$, where $y_\tau = x_{\tau+1}$ and consider the same loss function, the negative log-likelihood.

For TCN, there are three hyperparameters that we can tune for TC layers: the number of RESBLOCK $n_R$, the number of kernel filters $n_k$, and the size of kernels $k$. In the following
experiments, we vary these hyperparameters as \( n_R = \{3, 4, 5, 6, 7, 8\}, n_k = \{4\}, \) and \( k = \{2, 4, 8\} \). Note that we employ the same number of kernel filters for all \texttt{RESBLOCK}s (e.g., if \( n_k = 4 \), all TC layers in all \texttt{RESBLOCK}s have 4 kernel filters) because we observe that changing this parameter has a negligible impact to the performances in the near-term %ILI prediction. Lastly, as in the previous experiments, we vary lengths of historical observations \( w_{\text{hist}} = \{16, 32, 64, 128\} \).

For the choices of the number of residual blocks \( n_R = \{3, 4\} \), there seem to have no clear trend or differences between the performances of TCNs for all considered hyperparameters, \( k = \{2, 4, 8\} \) and \( w_{\text{hist}} = \{16, 32, 64, 128\} \). For \( n_R = \{5, 6, 7, 8\} \), the TCNs perform better with longer subsequences \( w_{\text{hist}} = \{64, 128\} \). We believe this is caused by the fact that increasing number of \texttt{RESBLOCK} results in larger dilation factors \( d \) and, consequently, larger receptive fields, which does not have a significant impact on shorter subsequences (i.e., \( w_{\text{hist}} = \{16, 32\} \)), but have a significant impact on longer subsequences (i.e., \( w_{\text{hist}} = \{64, 128\} \)).

Fig. 3-4 shows the results obtained by using the TCNs for several residual block counts, \( n_R = \{5, 6, 7, 8\} \) with the kernel size \( k = 4 \). The figures essentially show that RMSE, MAPE, and L2E tend to decrease and PCORR tend to increase as \( w_{\text{hist}} \) increases. From the experiments with different kernel sizes, \( k = \{2, 8\} \), and \( n_R = \{5, 6, 7, 8\} \), we observe the same trend (i.e., improved performances with longer \( w_{\text{hist}} \)) in most cases.

![Fig 3-4. Four performance indicators for one week-ahead predictions obtained with TCNs with varying numbers of residual blocks \( n_R = \{5, 6, 7, 8\} \) and kernel size \( k = 4 \).](image-url)
Next, we present results obtained with SNAIL. Again, we consider the same supervised learning setting for the input and the target data, i.e., the $i$-th input data sequence is $\{x^{(i)}_\tau\}_{\tau=1}^{w_{\text{hist}}}$ and the $i$-th target data sequence is $\{y^{(i)}_\tau\}_{\tau=1}^{w_{\text{hist}}}$, where $y_\tau = x_{\tau+1}$.

We follow the same network architecture considered in the original paper \cite{33}; there are three \textsc{AttentionBlocks} interleaved with two \textsc{TCBlocks} followed by a $1 \times 1$ convolutional layers. For each \textsc{TCBlock}, there are three hyperparameters for \textsc{TC} layers, the number of \textsc{TC} layers $n_{\text{TC}}$, the number of kernel filters $n_k$, and the size of kernels $k$; we consider the same choices of the hyperparameters for the two \textsc{TCBlocks}. From the empirical experience with the previous TCN experiments, we choose the values of the hyperparameters as follows: $n_{\text{TC}} = \{6, 7, 8\}$, $n_k = \{4, 8\}$, and $k = \{2, 4, 8\}$. For each \textsc{AttentionBlock}, there are two hyperparameters, the size of the attention keys $n_{\text{key}}$ and the size of the attention values $n_{\text{value}}$. We vary the value of the first parameter as $n_{\text{key}} = \{16, 32\}$ and fix the value of the second parameter to $n_{\text{value}} = 32$.

As shown in the experimental results of TCNs, the SNAIL architectures tend to perform better with longer subsequences (i.e., $w_{\text{hist}} = \{64, 128\}$). Fig. 3-5 depicts the results obtained by using the SNAIL architectures with the kernel size $k = 4$ for varying $n_{\text{TC}}$ and $n_{\text{key}}$, and shows that the best performance is achieved with $w_{\text{hist}} = 128$ in most cases (with the exception depicted in the yellow curve). We also observe that the SNAIL architectures with $k = \{2, 8\}$ and $n_k = 8$, of which experimental results are not shown, perform the best with $w_{\text{hist}} = 128$ in most cases.

### 3.3.5. Performance comparisons between Seq2Seq, TCN, and SNAIL

We now compare the three sequence models discussed above: Seq2Seq architecture with GRU cells, TCN, and SNAIL. For each performance metric, we pick the best performing neural network configurations (i.e., network architectures and their hyperparameters). Table 3-1 lists the neural network configurations of each considered neural networks that perform the best for each performance metric. Note that for RMSE and L2E, the same neural network configurations are used to achieve the best performances.

<table>
<thead>
<tr>
<th>Seq2Seq</th>
<th>TCN</th>
<th>SNAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSE $(n_k, w_{\text{hist}}) = (3, 128)$</td>
<td>$(n_k, w_{\text{hist}}) = (3, 128)$</td>
<td>$(n_{\text{TC}}, k) = (7, 4, 4, 32)$</td>
</tr>
<tr>
<td>MAPE $(n_k, w_{\text{hist}}) = (4, 64)$</td>
<td>$(n_k, w_{\text{hist}}) = (4, 64)$</td>
<td>$(n_{\text{TC}}, k) = (6, 2, 8, 32)$</td>
</tr>
<tr>
<td>L2E $(n_k, w_{\text{hist}}) = (3, 128)$</td>
<td>$(n_k, w_{\text{hist}}) = (3, 128)$</td>
<td>$(n_{\text{TC}}, k) = (7, 4, 4, 32)$</td>
</tr>
<tr>
<td>PCORR $(n_k, w_{\text{hist}}) = (5, 128)$</td>
<td>$(n_k, w_{\text{hist}}) = (5, 128)$</td>
<td>$(n_{\text{TC}}, k) = (7, 4, 4, 32)$</td>
</tr>
</tbody>
</table>
Fig 3-6 reports the four performance metrics measured in each state using Seq2Seq, TCN, and SNAIL. The states in Figs 3-6a–3-6d are presented in a decreasing order w.r.t. the relative performance of SNAIL against Seq2Seq (i.e., \( \frac{\text{Performance of SNAIL}}{\text{Performance of Seq2Seq}} \)). Thus, the relative performance of SNAIL against Seq2Seq decreases going from left to right.

Table 3-2 and Figs. 3-7 to 3-8 provide more information on the bar plots. First, Table 3-2 shows the four performance metrics measured by using the best performing neural network configurations (Table 3-1). Each performance metric is measured for each state and then averaged over 49 states. For all four performance metrics, both TCN and SNAIL outperform Seq2Seq; SNAIL performs slightly better than TCN. Recall that MAPE and L2E are relative measures and MAPE is a percentage. RMSE is an absolute measure, where the input data in the test set lies in \([0, 16]\).

Figure 3-7 depicts the boxplots of all four performance indicators corresponding to Seq2Seq, TCN, and SNAIL. These boxplots display statistical quantities from the results shown in Fig. 3-6: means, medians, and interquartiles ranges. Fig. 3-8 reports the proportions of the states where
TCN and SNAIL outperform Seq2Seq, respectively, and SNAIL outperforms TCN in 0, 1, 2, 3, or all 4 error indicators (RMSE, MAPE, L2E, and PCORR). Both TCN and SNAIL outperform Seq2Seq in 29 and 30 states for at least three (out of four) performance indicators. SNAIL performs only slightly better than TCN. These observations agree with the statistics reported in Table 3-2 and Fig. 3-7.

Finally, Figs. 3-9–3-11 illustrate the original %ILI and the predictions made by Seq2Seq and SNAIL for two seasons: 2017–2018 (validation) and 2018–2019 (test). To increase the legibility of the plots, %ILI predictions made by TCN are not reported. We note that the %ILI curves of
Table 3-2. The best performances achieved by Seq2Seq, TCN, and SNAIL. The performance metrics are averaged over 49 states. The symbols ↓ and ↑ indicate that the lower and higher values correspond to the better performance.

<table>
<thead>
<tr>
<th></th>
<th>Seq2Seq</th>
<th>TCN</th>
<th>SNAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSE (↓)</td>
<td>0.5756</td>
<td>0.5543</td>
<td><strong>0.5541</strong></td>
</tr>
<tr>
<td>MAPE (↓)</td>
<td>11.14</td>
<td>10.98</td>
<td><strong>10.86</strong></td>
</tr>
<tr>
<td>L2E (↓)</td>
<td>0.2012</td>
<td>0.1955</td>
<td><strong>0.1937</strong></td>
</tr>
<tr>
<td>PCORR (↑)</td>
<td>0.9104</td>
<td>0.9157</td>
<td><strong>0.9161</strong></td>
</tr>
</tbody>
</table>

TCN are very similar to the ones of SNAIL.

3.4. N-weeks-ahead predictions

Lastly, we perform experiments with varying prediction horizons \( w_{\text{pred}} \). We again consider Seq2Seq, TCN, and SNAIL models. For training Seq2Seq model, as in the previous Seq2Seq setting, the \( i \)-th input sequence is denoted by \( x_1^{(i)}, \ldots, x_{w_{\text{hist}}}^{(i)} \). The encoder RNN network processes this input sequence to produce a context vector, then the decoder receives the last element in the input sequence (i.e., \( x_{w_{\text{hist}}}^{(i)} \)) and the context vector to produce output \( \{o_1^{(i)}, \ldots, o_{w_{\text{pred}}}^{(i)}\} \), which attempts to match \( \{y_1^{(i)}, \ldots, y_{w_{\text{hist}}}^{(i)}\} = \{x_{w_{\text{hist}}+1}^{(i)}, \ldots, x_{w_{\text{hist}}+w_{\text{pred}}}^{(i)}\} \). For TCN and SNAIL, \( i \)-th input data sequence is \( \{x_{\tau}^{(i)}\}_{\tau=1}^{w_{\text{hist}}} \) and the \( i \)-th target data sequence is \( \{(y_{\tau}^{(i)}, \ldots, y_{\tau+w_{\text{pred}}-1}^{(i)})\}_{\tau=1}^{w_{\text{hist}}} \), where \( y_{\tau} = x_{\tau+1} \).

We conduct experiments for varying prediction horizons \( \{1, 2, 3, 4\} \) because the weekly %ILI report often takes one to four weeks to be processed and present the results in Table 3-3. For Seq2Seq, TCN, and SNAIL models, we pick a single configuration of hyperparameters for each model:

- for Seq2Seq, \( (n_b, w_{\text{hist}}) = (3, 128) \),
- for TCN, \( (n_R, k, n_k, w_{\text{hist}}) = (8, 4, 4, 128) \),
- for SNAIL, \( (n_{TC}, k, n_k, n_{key}, n_{value}, w_{\text{hist}}) = (7, 8, 4, 32, 16, 128) \),

and measure all four performance indicators with the three network models.

In most cases, both TCN and SNAIL outperform Seq2Seq in all error metrics for \( w_{\text{pred}} = \{1, 2, 3\} \). For \( w_{\text{pred}} = 4 \), SNAIL outperforms Seq2Seq in three error metrics and TCN for all error metrics and Seq2Seq outperforms TCN in all four error metrics. The prediction accuracy measured via using TCN and SNAIL in all four metrics are degraded as \( w_{\text{pred}} \) increases from 3 to 4. Interestingly, as opposed such degradation, there is virtually no degradation in the prediction accuracy measured via using Seq2Seq as \( w_{\text{pred}} \) from 3 to 4.
Fig 3-7. The boxplots of all four performance metrics measured by using Seq2Seq, TCN, and SNAIL. The boxplots summarize statistics of the data shown in Fig 3-6. The middle black and yellow lines indicate medians and means, respectively, and the boxes indicates the interquartile ranges. Top half: the entire range of the boxplots. Bottom half: the IQR of the boxplots excerpted.
3.5. Discussion

The series of experiments above show the effect of the simple, uni-modal, if noisy, influenza curves. Fig. 3-2 shows that adding multiple layers to the RNNs (irrespective of whether LSTM or GRU cells are used) only marginally improve predictive skill and training the model on a longer sequence of data (Fig. 3-1) does not result in any improvement. That is, the RNN models achieve their predictive skill with modest data and modest architectural complexity, and training a higher capacity model results in only a marginally better one week-ahead forecast. Fig. 3-3 shows the impact of different architectural choices (i.e., encoder-decoder type network with RNN-GRU decoder) and training data size for the Seq2Seq model, and we see that the improvement in predictive skill is modest: the Seq2Seq model performs only slightly better than the RNN models, implying that the complications of the Seq2Seq model is largely lost on predictive skill. We
Fig 3-9. Example plots of %ILI and predictions made by Seq2Seq and SNAIL for two seasons 2017-2018 (validation) and 2018-2019 (test). The original %ILI is depicted in solid black lines and the prediction results of Seq2Seq and SNAIL are depicted in solid red lines and dashed green lines.

believe that introducing more advanced deep learning techniques such as attention mechanism to Seq2Seq architecture is a key to make improvements in prediction accuracy as shown in Refs. [31, 51].
The TC-layer-based models also show better outcomes in Fig. 3-4 and 3-5—combinations of higher architectural complexity (i.e., $n_R > 6$ and $n_{TC} > 7$) and longer training sequence $w_{hist} > 32$ result in a better model. Fig. 3-6 and Table 3-2 show that, on the whole, TCN and SNAIL outperforms Seq2Seq. Table 3-3 shows a more complex behavior where both TCN and SNAIL...
Fig 3-11. Example plots of %ILI and predictions made by Seq2Seq and SNAIL for two seasons 2017-2018 (validation) and 2018-2019 (test). The original %ILI is depicted in solid black lines and the prediction results of Seq2Seq and SNAIL are depicted in solid red lines and dashed green lines.

Table 3-3. The performance metrics measured by Seq2Seq, TCN, and SNAIL for varying prediction horizons \{1, 2, 3, 4\}. The same sets of network hyperparameters for each model are used for varying horizons. The downward/upward arrows in parentheses indicate smaller/larger values are preferred.

<table>
<thead>
<tr>
<th>horizon</th>
<th>(\text{RMSE (\downarrow)})</th>
<th>(\text{MAPE (\downarrow)})</th>
<th>(\text{L2E (\downarrow)})</th>
<th>(\text{PCORR (\uparrow)})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Seq2Seq</td>
<td>0.5756</td>
<td>0.8453</td>
<td>1.037</td>
<td>1.052</td>
</tr>
<tr>
<td>TCN</td>
<td>0.5543</td>
<td>0.8117</td>
<td>0.9515</td>
<td>1.155</td>
</tr>
<tr>
<td>SNAIL</td>
<td>0.5617</td>
<td>0.7915</td>
<td>0.9696</td>
<td>1.048</td>
</tr>
<tr>
<td>Seq2Seq</td>
<td>0.2012</td>
<td>0.2850</td>
<td>0.3575</td>
<td>0.3511</td>
</tr>
<tr>
<td>TCN</td>
<td>0.1955</td>
<td>0.2790</td>
<td>0.3237</td>
<td>0.3851</td>
</tr>
<tr>
<td>SNAIL</td>
<td>0.1956</td>
<td>0.2667</td>
<td>0.3319</td>
<td>0.3479</td>
</tr>
</tbody>
</table>

perform better than Seq2Seq although Seq2Seq demonstrated less degradation of accuracy as the prediction horizon increases from 3 to 4. Overall, we conclude that the TC-layer-based models achieve better performance due to its larger receptive field, which allows the models to take in and process longer training sequences easily.

**Related Work**

Various deep learning techniques for sequence modeling have been employed for ILI predictions. As we described in Introduction, nearly all deep-learning-based approaches heavily rely on RNNs...
with LSTM. We categorize these approaches into two classes: one class of approaches that utilizes only CDC data and another class of approaches that utilizes external factors affecting flu activity or external information that can be used as an indicator of flu activity.

**CDC data only**

In Ref. [52], weekly influenza activity levels at week $\tau$ were used as an input to convolutional layers to extract features that capture spatial correlations across regions where the statistics are collected (e.g., the U.S. states) and then the extracted features were used as an input to an RNN with GRU to capture temporal correlations of the weekly influenza activity levels. Ref. [56] studied four variant constructions of LSTM networks with %ILI data: one instance is a single LSTM network, where a single application of the trained network gives multi-step ahead predictions, and another instance is a set of multiple LSTM networks, where each LSTM is trained to predict %ILI at specific time index of multi-step ahead predictions.

**External factors**

Another approach to modeling ILI data employs time-sequence models augmented with auxiliary information. Ref. [47] utilized features extracted from a Twitter dataset as input features to LSTM networks along with ILI information collected from military populations. Ref. [46] used climate information and geo-spatial factors as input features to LSTM networks, while Ref. [49] proposed a framework for high-resolution (e.g., county-level) ILI predictions, which generates a high-resolution synthetic dataset via an epidemic simulator and then use the dataset for training a neural network. The proposed neural network consists of two LSTM networks; one processes within-season sequences and the other processes between-season sequences. In Ref. [31], a Seq2Seq model with LSTM is trained with %ILI and Google trends data.
4. CONCLUSION

In this study, we have investigated the feasibility of modeling epidemiological data with variants of temporal convolutional network models, and their performance gain over recurrent neural network models. We investigate whether the nonlinear transformations that deep learning models allow provide any advantage in forecasting skill over linear methods and perform extensive experiments for comparing all considered neural network models including RNN-GRU, Sequence-to-Sequence with RNN-GRU decoder and neural ordinary differential equation decoder, temporal convolution networks, and simple neural attentive meta-learner. We have observed that RNN-LSTM, which was shown to be far better than conventional ARIMA models as in Ref. [51], performs the worst among all considered neural network models and we find that neural networks based on temporal convolutional layers (TCN and SNAIL) tend to outperform RNN-LSTM/RNN-GRU and Seq2Seq models.

However, considering significant increase in neural network complexity, the performance improvements made by TCN and SNAIL over RNN-GRU can also be seen as modest, which suggests that modeling the simple, uni-modal, if noisy, form of influenza curves may not require a tremendous degree of complexity in the neural network architectures. This bottleneck may be due to an intrinsic limitation of purely data-driven approach and can be overcome by using external data or indicators of influenza-like illness or building a model that combines a data-driven approach with mechanistic or compartmental models. Nonetheless, we observe from our extensive experiments that complex deep learning models can be fitted to relatively modest epidemiological data without suffering from over-fitting thanks to a mini-batching stochastic gradient optimizer with an early-stopping strategy, and can provide better forecasts than conventional data-driven models.
REFERENCES


APPENDIX

Long Short-Term Memory

Fig. 4-1 illustrates the diagram of a LSTM cell. Below we describe the canonical set of operations for this architecture.

**Forget gate** The forget gate controls the fraction to which internal cell state is let in from the previous step. The forget gate takes the input data $x_\tau$ and the hidden state $h_{\tau-1}$ from the previous cell as the input and then produces a value between $[0, 1]$ with the element-wise sigmoid nonlinear function $\sigma(\cdot)$ such that

$$f_\tau = \sigma(b^f + U^f x_\tau + W^f h_{\tau-1}),$$

where $b^f$ representing biases and $U^f$ and $W^f$ the weights of the forget gate.

**Input gate** The input gate controls the extent of which the new input data $x_\tau$ contributes to updating the LSTM cell state. The input gate also takes the hidden state $h_{\tau-1}$ from the previous cell as the input and then produces a value between $[0, 1]$ with the element-wise sigmoid nonlinear function $\sigma(\cdot)$ such that

$$g_\tau = \sigma(b^g + U^g x_\tau + W^g h_{\tau-1}),$$

where $b^g$ representing biases and $U^g$ and $W^g$ the weights of the forget gate.

Together with the forget gate and the input gate, the new cell state $s_\tau$

$$s_\tau = f_\tau \odot s_{\tau-1} + g \odot \sigma (b + U x_\tau + W h_{\tau-1}),$$  \hspace{1cm} (4.0.1)
where $b, U,$ and $W$ are biases and weights to the LSTM cell and $\odot$ denotes element-wise multiplications. The MLP block stands for the second operand of the second term on the right-hand side (4.0.1).

**Output gate** The output gate controls the extent to which the information flows into the output using a similar gating mechanism

$$q_\tau = \sigma(b^q + U^q x_\tau + W^q h_{\tau-1}),$$

where $b, U^q$, and $W^q$ are biases and weights of the output gate, resulting in $q_\tau \in [0, 1]$. The hidden state of the LSTM cell is then computed as

$$h_\tau = \tanh(s_\tau) \odot q_\tau.$$

**Gated Recurrent Unit**

GRU does not have an additional cell state as in LSTM and consists of the two gating units, the update gate and the reset gate. Fig 4-2 illustrates the diagram of GRU.

![GRU cell diagram](image)

The output of the GRU is computed as follows:

$$h_\tau = u_{\tau-1} \odot h_{\tau-1} + (1-u_{\tau-1})\sigma(b + Ux_\tau + W(r_{\tau-1} \odot h_{\tau-1})), \quad (4.0.2)$$

where the values of the update gate and the reset gate are determined with similar expressions as for the LSTM architecture

$$u_\tau = \sigma(b^u + U^ux_\tau + W^uh_\tau)$$

and

$$r_\tau = \sigma(b^r + U^rx_\tau + W^rh_\tau).$$

Analogous to the LSTM diagram, the MLP block stands for the second operand of the second term on the right-hand side (4.0.2). The “1-” block computes $1 - u_{\tau-1}$. 

40
Comparison between LSTM and GRU

The GRU architecture is simpler compared to LSTM; it does not have the cell state and only two gates. The update gate combines the role of the forget gate and the input gate, which results in simpler recurring network architectures with less network parameters. Thus, LSTM performs better in many applications, not just large-scale applications [7], but also in rather simple applications [50]. This does not mean, however, that LSTM always performs better than GRU. In certain tasks, including natural language processing, speech signal modeling, GRU performs similar to LSTM [39], or even performs better in small datasets [13].

Temporal Convolutional Network

We describe here the sequence of operations in a TCN. Let us denote one-dimensional input sequence by $x \in \mathbb{R}^n$ and the kernel filter by $f \in \mathbb{R}^k$.

Causal convolution The causal convolution is the operator that can only look at the current time and previous input data, i.e. for $t \leq \tau$. This can be formally written as

$$(x \ast f)(\tau) = \sum_{i=0}^{k-1} f_i \cdot x_{\tau-i},$$

where $\ast$ defines the convolution between two sequences.

Dilated causal convolution The dilated causal convolution increases the size of receptive field by adding dilation factor $d$:

$$(x \ast_d f)(\tau) = \sum_{i=0}^{k-1} f_i \cdot x_{\tau-d\cdot i}$$

The convolution operator $\ast_d$ takes every $d$ element of the input $x$. Thus, a series of dilated causal convolutions with increasing dilation factor $d = 2^l$ increase the receptive field exponentially.

RESBLOCK TCN employs the residual block consisting of two layers of dilated causal convolutions, weight normalization, ReLU, and Dropout, and an identity mapping from the input. Formally, RESBLOCK can be written as

$$z_{i+1} = z_i + \mathcal{F}(z_i),$$

where $z_i$ and $z_{i+1}$ are the input and the output of RESBLOCK, and $\mathcal{F}(\cdot)$ is a function consisting of the two layers of dilated causal convolutions.

Fig [4-3] illustrate the flow of computations performed in one residual block. In the residual block, two dilated convolutional layers share the same hyperparameters (i.e., kernel size $k$, dilation factor $d$, and the number of channels $n_k$). By stacking up the residual blocks with increasing dilation factors $d$, the size of the receptive field can grow exponentially.
The residual block consists of two layers of dilated convolutional layers, each of which followed by weight normalization, ReLU, and Dropout. Then the resulting quantity of the dilated convolutions and the input are summed element-wise. If the input and the output sizes do not match, an optional $1\times1$ convolution can be used.

**Simple Neural Attentive Meta Learner**

The detailed operations of SNAIL are described below

**TCBLOCK** The main component of TCBLOCK is also the one-dimensional dilated causal convolutions. TCBLOCK consists of a series of multiple DENSEBLOCKS with increasing dilation factors. The number of DENSEBLOCKS in one TCBLOCK is determined by the length $n$ of the input sequence (i.e., $[n]$). That is, TCBLOCK ensures that the size of receptive field covers the entire input sequence. The $\ell$th DENSEBLOCK consists of dilated causal convolutions with dilation factors $2^{\ell-1}$. In the original paper, the dilation factor for the $\ell$th DENSEBLOCK is $2^\ell$. The DENSEBLOCK consists of two parallel dilated causal convolutions with the same hyperparameter settings (i.e., kernel size $k$, dilation factor $d$, and the number of channels $n_k$), but followed by different nonlinear activations tanh and sigmoid. The outputs of the convolutional layers are multiplied element-wise and then the resulting quantity is concatenated with the input. Fig 4-4 illustrates the computation flow of DENSEBLOCK.

**ATTENTIONBLOCK** The ATTENTIONBLOCK is designed to perform the key-value-pair-based self-attention mechanism proposed by [45]. The self-attention mechanisms refer to a mechanism that learns relations of elements in different positions of a single sequence. The self-attention mechanism employed in SNAIL is a soft-attention mechanism because the relations of entire elements in a sequence are considered. This is in contrast to a hard-attention mechanism, where the model attends to a specific region (e.g., a small patch in an image).
Fig 4-4. SNAIL DENSEBLOCK. DENSEBLOCK consists of two parallel dilated causal convolutions. The outputs of the convolutional layers are multiplied element-wise and then the resulting quantity is concatenated with the input.

The key-value-pair-based attention mechanism consists of three quantities: key, value, and query. When query is given, the attention mechanism first attempts to match key, which is associated with a value that is closest to what the query is looking for. In SNAIL, all three quantities are computed by applying a single-layer feed-forward network to the input of ATTENTIONBLOCK, which is usually the hidden states produced by the TCBLOCK. ATTENTIONBLOCK finds how relevant query and key are by using the scaled-dot product \( \frac{QK^T}{\sqrt{n_{\text{key}}}} \); by applying causally masked softmax to convert the resulting quantity into the probability. Here, the causal mask prevents the \( \tau \)-th query cannot have access to future key/values, i.e., \( P = \text{softmax} \left( M \odot \frac{QK^T}{\sqrt{n_{\text{key}}}} \right) \), where \( M \) is the causal mask whose upper triangular part is zeroed out. Then multiply \( P \) with \( V \) finally gives values that queries look for. Fig 4-5 illustrate the ATTENTIONBLOCK computation flow.

Fig 4-5. SNAIL DENSEBLOCK. DENSEBLOCK consists of two parallel dilated causal convolutions. The outputs of the convolutional layers are multiplied element-wise and then the resulting quantity is concatenated with the input.