Foreclassing: A new machine learning perspective on human decision making with temporal data

Daniel Andrew Coulson
Department of Statistics and Data Science
Cornell University
Ithaca, NY 14850

Martin T. Wells
Department of Statistics and Data Science
Cornell University
Ithaca, NY 14850

Abstract

We propose a novel machine learning problem, which we call the 'Foreclassing problem', where we wish to make a time series classification decision, but the loss of our decision is observed at a future time point and the true class label is dependent on the past realizations of the time series and future observations. We motivate this problem from the daily decisions humans make and through a decision theoretic theorem. We formalize the problem mathematically and propose a novel neural network architecture with two new types of neural network layers to solve it, directing our model to process information in a manner akin to human decision making, as demonstrated in the neuroscience literature. We then show how our proposed approach and architecture achieves comparable, to superior, performance when compared to existing state of the art time series classification methods through simulation studies and real-world examples.

1 Introduction

Humans face daily instances of making classification decisions based on temporal data where the class label is only revealed several time points after a classification decision must be made, which we call the 'Foreclassing problem'. For example, when professors are reviewing PhD applications, they may consider each student's past course work and research experience, together with any additional information. Based on this, they will then predict how a student will perform during the PhD program if they are admitted, and then subsequently accept or reject their application. However, the professor will only know if their decision to accept a student is correct three to four years into the PhD program. More broadly, Foreclassing is related to predictive coding in the neuroscience literature, such as Blom et al. [2020], Hogendoorn and Burkitt [2018]. For example, if a person wishes to catch a ball, their brain must firstly predict the future location of the ball as it takes time for the brain to process the current input sensory information. The brain, based on its own predictions of the future and using the sensory input of the past, will then decide on an appropriate action to take. If the brain did not predict the future location of the ball because of the lag in processing sensory information from the moment it is received, the person would likely fail to catch the ball. Beyond neuroscience, Foreclassing has connections to other areas too, such as predictive control of models and robotics, see Doganis et al. [2008],Lenz et al. [2015], and Zhang et al. [2020]. In predictive control we wish to optimize our decisions based on the forecasted future states of a dynamical system, and in robotics it is helpful for a robot to have forecasts of the future so that they can adjust their actions accordingly, for example changing their grasp of an object they are holding. Other applications include ECG classification, where we wish to classify patients with different heart conditions based upon ECG time series; or in Finance, where a trader is trying to decide whether to buy a stock in the hope that the price will increase in the future, based upon historic stock price time series.

Preprint. Under review.

As Foreclassing involves making classification decisions based upon future events, there is inherent uncertainty, hence notions of uncertainty should be incorporated into our models. A popular approach to provide notions of uncertainty is Bayesian statistics, which assumes parameters of interest are random and have an associated non-degenerate probability distribution; with existing neuroscience literature - such as the review by Knill and Pouget [2004] - linking Bayesian statistics to neural computation. Therefore, we propose to use a novel Bayesian convolutional neural network (LeCun et al. [1989], Gal et al. [2017]) to solve the Foreclassing problem, which we call 'ForeClassNet'. Humans incorporate notions of uncertainty into their decision making, but unfortunately typical Bayesian analysis only performs uncertainty quantification after the model has been fitted. However, for a network solving the Foreclassing problem, we want it to have some quantification of its own uncertainty about the future before making its classification decision, not the posterior variance after the model has arrived at its classification decision. To provide ForeClassNet with such an ability, we propose a new neural network layer called 'Welford mean-variance (WMV) layers', which updates ForeClassNet's estimates of its means and variances of its forecasts of future observations through successive forward passes of ForeClassNet using Welford's online algorithm (Welford [1962]).

Within ForeClassNet we utilize a convolutional layer which is made up of hand-crafted convolutional filters which represent fundamental aspects of a time series, increasing, decreasing, and peak patterns (Ismail-Fawaz et al. [2022]).However, hand-crafted filters are nonprobabilistic as their weights are fixed, which is not in the spirit of a Bayesian network. More generally, for learnable convolutional filters, although we can assign prior distributions to the weights within a learnable convolutional filter we are still uncertain as to the length that the convolutional filter should be. We can solve both problems by assigning a prior distribution to the possible filter lengths. This leads us to propose 'Boltzmann convolutional (BC) layers'. BC layers firstly construct a filter length set consisting of all the lengths we want to consider. We allow these to be drawn according to a discrete Boltzmann distribution where the logits in the distribution associated with each filter length are learnable by the network. Then within each BC layer we train several sub-convolutional layers, each consisting of several convolutional filters with the same filter length. The BC layer then convolves the input for the layer with each of these sub-convolutional layers and takes a learnable convex combination of the outputs, where the weights associated with the output of each sub-convolutional layer are given by the probabilities of their associated filter lengths according to the Boltzmann distribution. This also provides the model with an ability to perform soft model selection and provide multi-temporal resolution representations of the input time series. In our network we utilize BC layers with hand crafted convolutional filter weights where only the distribution of the filter lengths are learnable, and BC layers where both the convolutional filter weights and the distribution of the filter lengths are learnable.

Related work Foreclassing can be seen as a generalization of the time series classification (TSC) problem (see Middlehurst et al. [2024b] for a review of TSC) which has not been discussed in the literature. The closest existing literature to Foreclassing is Liu et al. [2014] who classify different time series by using estimated forecast densities. This is closer to typical TSC as the class label is revealed at the moment the decision is made instead of it only being revealed after we have observed the future data; they are classifying models of time series which have similar forecast densities, rather than the true label being a function of the observations at past and future time points after the decision has been made as is the general case of the Foreclassing problem. Another related problem is early time series classification (eTSC), for example Schäfer and Leser [2020], where we want to classify a time series as soon as possible, that is with as few time points as possible, which is different to Foreclassing as in eTSC the class label is only a function of the observed time series, not the future observations. There are also methods related to BC layers in the literature, such as Chen et al. [2020] which have linear combinations of filters of the same length with the combination weights being input dependent and computed using a squeeze and excitation (Hu et al. [2018]) approach. Wang et al. [2021] build upon this by allowing for different filter lengths. Our approach is mathematically simpler, allowing for uncertainty quantification through probabilistic filter lengths. The multiple probabilistic filter lengths help in problems such as ECG classification (Table 4, Section 7) due to the non-stationarity of the time series. Han et al. [2018] consider a continuous definition of filter size and use a linear interpolation of the closest integers that lower and upper bound the estimated continuous filter size. BC layers allow for convex combinations of any positive integer-valued filter lengths; they have learnable combination weights rather than the weights simply being a function of the lengths; and have a probabilistic interpretation of the filter lengths. Welford's algorithm has

been used before, usually for normalization purposes such as Vasan et al. [2024]; but WMV layers explicitly incorporate the mean and variance of given quantities of interest outputted from the model as input to deeper layers during forward propagation.

Our contributions We propose a new machine learning problem called Foreclassing, formalize this problem mathematically, and prove a motivating decision theoretic theorem. We also construct a new neural network architecture called ForeClassNet which involves forming two new neural network layers, Boltzmann Convolutions and Welford mean-variance layers.

2 Mathematical formulation of Foreclassing

As this is a new machine learning problem we firstly formally define the problem we are proposing which we call the 'Foreclassing problem'.

Foreclassing problem statement Suppose we have N observed time series each of length m, $\boldsymbol{x}^{(i)} = (x_1^{(i)}, ..., x_m^{(i)})^T$ for $i = 1, ..., N$. Each time series is a member of one of L classes, where the class in which a time series is in is realized only in the future, say k time points ahead. Denote future observations of the time series by $\boldsymbol{x}_*^{(i)} = (x_{m+1}^{(i)}, ..., x_{m+k}^{(i)})^T$ and the class time series i is by $y^{(i)} \in \{0, 1, ..., L-1\}$, for $i = 1, ..., N$. Then the class label of time series i is given by $y^{(i)} = f(\boldsymbol{x}^{(i)}, \boldsymbol{x}_*^{(i)}) = f(\boldsymbol{x}^{(i)}, w(\boldsymbol{x}^{(i)}), \epsilon_{m+1}, \epsilon_{m+2}, ..., \epsilon_{m+k})$, the function w is a mapping from the observed time series to future observations of the time series,$\epsilon_{m+1}, ..., \epsilon_k$ represents the uncertainty associated with future outcomes that are unobserved, and f is a mapping from the observed and future time series to the class to which a given time series belongs.

This results in a multitask learning problem for which we aim to estimate the functions $f(\cdot)$ and $w(\cdot)$ by using a deep Bayesian neural network and minimizing the categorical cross-entropy loss function and mean squared error loss function, respectively. Foreclassing is different from standard time series classification that would only be estimating the function $f(\cdot)$ which only considers the observed time series. However, as demonstrated by the Foreclassing Theorem below, solving only the TSC problem, that is only considering the observed time series, can lead to increased classification error.

Foreclassing Theorem Consider the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and the random variables $\boldsymbol{X} : \Omega \to \mathbb{R}^m$, $\boldsymbol{X}_* : \Omega \to \mathbb{R}^k$, and $Y : \Omega \to \{0, 1\}$, where $Y = f(\boldsymbol{X}, \boldsymbol{X}_*)$ for some unknown measurable function $f(\cdot)$. Assume $\boldsymbol{X}_*$ is not fully determined by $\boldsymbol{X}$, that is, $\sigma(\boldsymbol{X}) \subsetneq \sigma(\boldsymbol{X}, \boldsymbol{X}_*)$. In addition, assume that $\boldsymbol{X}_*$ provides additional information regarding Y beyond $\boldsymbol{X}$ i.e.$\exists A \in \sigma(\boldsymbol{X}_*) \backslash \sigma(\boldsymbol{X})$ where $\mathbb{P}(A) > 0$ such that $\mathbb{P}(Y = y|\boldsymbol{X}, \boldsymbol{X}_*) \neq \mathbb{P}(Y = y|\boldsymbol{X})$. Then for any classifier $\tilde{f}(\boldsymbol{X})$ that ignores $\boldsymbol{X}_*$ there exists a classifier $\hat{f}(\boldsymbol{X}, \boldsymbol{X}_*)$ whose classification error is strictly smaller.

Proof of Foreclassing Theorem Consider the two Bayes optimal classifiers,

$$\tilde{f}_{\text{Bayes}}(\boldsymbol{X}) = \arg \max_{y \in \{0,1\}} \mathbb{P}(Y = y|\boldsymbol{X} = \boldsymbol{x})$$

and

$$\hat{f}_{\text{Bayes}}(\boldsymbol{X}, \boldsymbol{X}_*) = \arg \max_{y \in \{0,1\}} \mathbb{P}(Y = y|\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{X}_* = \boldsymbol{x}_*).$$

Their respective risks under the 0-1 loss are given by

$$R(\tilde{f}_{Bayes}(\boldsymbol{X})) = \mathbb{E}_{\boldsymbol{x} \sim \mu_{\boldsymbol{X}}}[1 - \max_{y \in \{0,1\}} \mathbb{P}(Y = y|\boldsymbol{X} = \boldsymbol{x})]$$

and

$$R(\hat{f}_{Bayes}(\boldsymbol{X}, \boldsymbol{X}_*)) = \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{x}_*) \sim \mu_{\boldsymbol{X} \times \boldsymbol{X}_*}}[1 - \max_{y \in \{0,1\}} \mathbb{P}(Y = y|\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{X}_* = \boldsymbol{x}_*)].$$

Since $\boldsymbol{X}_*$ contains additional information about Y beyond $\boldsymbol{X}$, we have

$$\max_{y \in \{0,1\}} \mathbb{P}(Y = y|\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{X}_* = \boldsymbol{x}_*) > \max_{y \in \{0,1\}} \mathbb{P}(Y = y|\boldsymbol{X} = \boldsymbol{x}).$$

This implies that

$$1 - \max_{y \in \{0,1\}} \mathbb{P}(Y = y|\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{X}_* = \boldsymbol{x}_*) < 1 - \max_{y \in \{0,1\}} \mathbb{P}(Y = y|\boldsymbol{X} = \boldsymbol{x}).$$

Taking expectations, we obtain

$$\mathbb{E}_{(\boldsymbol{x},\boldsymbol{x}_*)\sim\mu_{\boldsymbol{X}\times\boldsymbol{X}_*}}[1-\max_{y\in\{0,1\}}\mathbb{P}(Y=y|\boldsymbol{X}=\boldsymbol{x},\boldsymbol{X}_*=\boldsymbol{x}_*)] < \mathbb{E}_{\boldsymbol{x}\sim\mu_{\boldsymbol{X}}}[1-\max_{y\in\{0,1\}}\mathbb{P}(Y=y|\boldsymbol{X}=\boldsymbol{x})].$$

Thus, we can then conclude that

$$R(\hat{f}_{Bayes}(\boldsymbol{X},\boldsymbol{X}_*)) < R(\tilde{f}_{Bayes}(\boldsymbol{X})) \leq R(\tilde{f}(\boldsymbol{X})),$$

where $\tilde{f}(\boldsymbol{X})$ is any classifier that ignores $\boldsymbol{X}_*$.

3 Bayesian deep learning

In Bayesian statistics we have observed data $\boldsymbol{x}$ and a model to estimate the population level probability distribution associated with this data (assuming the densities exist) with probability density function $p(\boldsymbol{x}|\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the unknown (random) parameters in the model, and our goal is to find the posterior distribution with density $p(\boldsymbol{\theta}|\boldsymbol{x})$. By Bayes Theorem we have that

$$p(\boldsymbol{\theta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{x})}, \text{ where } p(\boldsymbol{x}) = \int_{\boldsymbol{\theta}} p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$$

where $p(\boldsymbol{\theta})$ is known as the prior distribution and represents a person's beliefs about the possible values of $\boldsymbol{\theta}$ before observing any data. Unless under special circumstances, such as conjugacy, the standard approach to Bayesian inference is Markov Chain Monte Carlo (MCMC) where we construct a Markov chain whose stationary distribution is the posterior distribution of interest. For large models, such as neural networks, MCMC can be computationally expensive so practitioners instead opt for variational inference (VI). VI aims to find a simple approximating distribution to the posterior known as the variational distribution which is found by solving $\min_{f(\boldsymbol{\theta})\in D} KL(f(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\boldsymbol{x}))$, where D represents the family of probability distributions we are optimizing over, and $KL(\cdot||\cdot)$ represents the KL divergence between two probability density functions. Usually the KL divergence is computationally intractable so instead we often maximize the evidence lower bound (ELBO) which is equivalent to minimizing the KL divergence. Particularly, ELBO$(f(\boldsymbol{\theta})) = \mathbb{E}_{f(\boldsymbol{\theta})}[log(p(\boldsymbol{\theta}|\boldsymbol{x})) - log(f(\boldsymbol{\theta}))]$.Gal and Ghahramani [2016] show that by applying dropout at both training and inference time in a deep neural network minimizing the cost objective is equivalent to minimizing the KL-divergence between an approximating distribution and the posterior distribution of a deep gaussian process (Damianou and Lawrence [2013]) - but this requires tuning of dropout probabilities. To avoid this (Gal et al. [2017]) propose a continuous relaxation termed 'concrete dropout' which allows the dropout probabilities to be learnable which we use in our Bayesian network.

4 Neural network architecture

4.1 Boltzmann convolutional layers

To construct a Boltzmann convolutional (BC) layer we first define a filter set z in advance. A filter set we found works well in our experiments was $z := \{3, 6, 12, 18, 24, 30, 36\}$ where the cardinality of z is $|z| = 7$ and for longer time series longer filter lengths may be preferred. We then assign a discrete Boltzmann probability distribution to the filter set. Particularly,

$$\mathbb{P}(\text{kernel filter length } = z_j) = softmax(\frac{1}{T}\boldsymbol{p})_j = exp\{\frac{p_j}{T}\} \div \sum_{i=1}^{|z|} exp\{\frac{p_i}{T}\}. \quad (1)$$

The vector $\boldsymbol{p}$ is the vector of energies (logits) and is trainable by the network. This allows the network to learn which filter lengths to focus on and perform soft model selection in a data driven manner. We also set $T = 1$, with T being absorbed into the vector of energies. We then construct several convolutional layers each associated with a particular filter length. Each convolutional layer consists of many convolutional filters with their filter lengths associated with the given convolutional layer. We then convolve each of these convolutional layers with the input x to obtain their output. These outputs are then combined according to (2),

$$\text{BC}(x) = \sum_{j=1}^{|z|} \mathbb{P}(\text{filter length} = z_j)[\text{convolutional layer with filter lengths equal to } z_j] * (x). \quad (2)$$

By taking the convex combination of the outputs of the individual convolutional layers we can consider multiple temporal resolutions of the time series, and perform soft model selection by learning the combination weights.

4.2 Welford mean-variance layers

Our network first forecasts the future observations of the input time series before making a classification decision. However, the future is uncertain and we want our model to take this uncertainty into account when it is making a classification decision instead of uncertainty quantification after it has made a classification decision. To allow for this we propose Welford mean-variance (WMV) layers for Bayesian neural networks. WMV layers iteratively update the mean and variance of the model's predictions of quantities of interest, through successive forward passes using Welford's algorithm (Welford [1962]), which can be used deeper in the network. In this work we use WMV layers to compute the means and variances for the model's forecasts of future observations of a given time series. Particularly for a given quantity q we update the network's estimate of its mean $\bar{q}_n$ and variance $s^2_{q,n}$ in the nth forward pass with the value of the quantity q_n produced by the model in the nth forward pass according to

$$\bar{q}_n = \bar{q}_{n-1} + \frac{q_n - \bar{q}_{n-1}}{n} \text{ and } s^2_{q,n} = (1 - \frac{1}{n}) s^2_{q,n-1} + \frac{(q_n - \bar{q}_{n-1})(q_n - \bar{q}_n)}{n}. \tag{3}$$

We use the population variance formula to update the network's estimate of its variance instead of the sample variance formula to avoid initialization problems during the first forward pass as otherwise we would suffer a division by zero error.

5 Hand crafted filters

Ismail-Fawaz et al. [2022] propose to use hand crafted filters for time series classification particularly an increasing filter, a decreasing filter, and a peak filter. We build upon this work by using hand-crafted filters for time series forecasting instead of classification, and using causal convolutions (van den Oord et al. [2016]) rather than forward looking convolutions. We also define a peak filter using Pascal's triangle from combinatorics rather than deriving a peak filter from the probability density function of a normal distribution. In particular,

$$\text{Decreasing filter} = [(-1)^{i+1}, \text{ for } 0 \leq i \leq h-1], \tag{4}$$

$$\text{Increasing filter} = [(-1)^{i}, \text{ for } 0 \leq i \leq h-1], \tag{5}$$

$$\text{Peak filter} = \begin{cases} -\frac{3}{h}\left[\left(\frac{h-3}{3}\right) C(i)\right], & \text{for } 0 \leq i \leq \frac{h-3}{3}, \\ \frac{6}{h}\left[\left(\frac{h-3}{3}\right) C\left(i - \frac{h}{3}\right)\right], & \text{for } \frac{h}{3} \leq i \leq 2\left(\frac{h-3}{3}\right) + 1, \\ -\frac{3}{h}\left[\left(\frac{h-3}{3}\right) C\left(i - 2\left(\frac{h-3}{3}\right) - 2\right)\right], & \text{for } 2\left(\frac{h-3}{3}\right) + 2 \leq i \leq h-1, \end{cases} \tag{6}$$

where nCk denotes the binomial coefficient formula for integers $0 \leq k \leq n$.

6 ForeClassNet

Figure 1 shows our proposed Bayesian neural network architecture ForeClassNet. We apply concrete dropout (Gal et al. [2017]) to each layer of ForeClassNet, except for the hand-crafted Boltzmann convolutional layer and Welford mean-variance (WMV) layer. We use causal padding and causal convolutions rather than forward looking convolutions to help preserve the temporal information from the input time series to encourage the network in forecasting. We train our model using the sum of the categorical cross entropy loss function, mean squared error loss function, and regularization terms for model weights and dropout probabilities. We also use the learned representation of the observed time series from the forecasting component of the network, from the time distributed dense layer, in the classification component of the network; with the motivation being that just as our hand-crafted filters represent important aspects of a time series, the learned representation of the time series from different tasks will also represent fundamental aspects of a time series. We then expand the dimension of this so we now have two columns; the first column contains the values of the time series under the learned representation of the network, and the second column contains zeros to inform the deeper components of the network that these data were observed and not predicted by

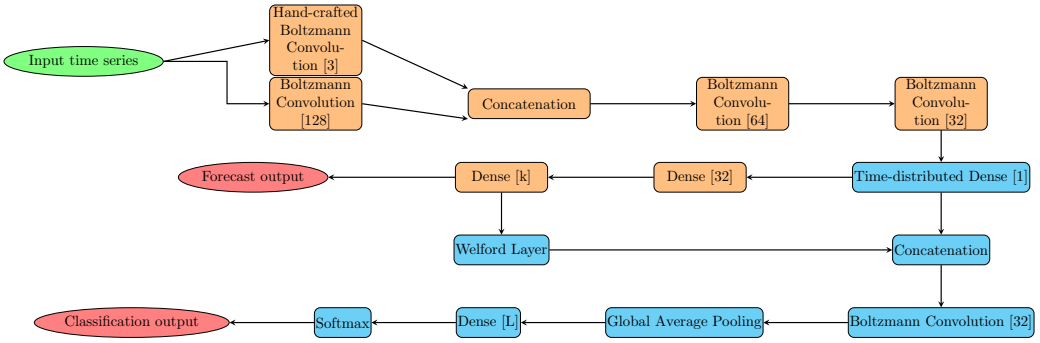

Figure 1: Graphic of our proposed network for time series forecasting and classification. The numbers in the brackets represent how many nodes are in a given layer, for example Dense [k] represents a dense layer with k nodes. The brackets in Boltzmann convolutional layers denote how many filters are in each sub-convolutional layer. We use a ReLU activation for most of the nodes, a Leaky ReLu with a slope of 0.01 in the Dense[32] layer, and the identity function in the time-distributed dense layer,Dense[k] layer, Dense[L] layer, and the Welford mean-variance layer.

the network, and therefore the model's variance in these predictions is zero. We then concatenate the forecasted observations' means and variances from the WMV layer to the end of the time series from the time distributed dense layer. That is the output of the final concatenation layer for a given time series will be

$$\begin{bmatrix} \phi_1 & \phi_2 & \dots & \phi_m & \bar{\phi}_{m+1} & \dots & \bar{\phi}_{m+k} \\ 0 & 0 & \dots & 0 & \hat{var}(\phi_{m+1}) & \dots & \hat{var}(\phi_{m+k}) \end{bmatrix}^T \tag{7}$$

where $\phi_1, ..., \phi_m$ are the learned representations of the observed time series, m is the number of time points where we have observations from the given time series, and the variance of each associated time point is set to zero. Subsequently for a given time series, $\bar{\phi}_{m+1}, ..., \bar{\phi}_{m+k}$ denotes the sample mean of the model's forecasts for the future observations over a forecast horizon of length k, and $\hat{var}(\phi_{m+1}), ..., \hat{var}(\phi_{m+k})$ denotes the sample variance of the model's forecasts for the future observations. For multivariate time series this is extended to a tensor, for example in our first experiment in Section 7 we have the x and y coordinates for 2 dimensional trajectories of projectiles and the resulting dimension of this object is [m+k,2,2]. By incorporating both forecasting,forecast uncertainty,and classification into our network it can help provide more explainable decision making with temporal data, akin to how humans process such data as discussed in the introduction. We also use dilated convolutions, starting with a dilation of 2 in the second Boltzmann convolutional layer and doubling it in each subsequent layer, in view of increasing the receptive field.

7 Experiments

We compare ForeClassNet to the two state of the art TSC methods as identified in Middlehurst et al. [2024b], specifically Multi Rocket Hydra (MR Hydra) and Hive-Cote 2 (HC2), see Dempster et al. [2023] and Middlehurst et al. [2021] respectively. Deep learning methods tend not to perform as well in TSC, but the best deep model is considered to be the Inception Time model (Ismail Fawaz et al. [2020]), an ensemble of five inception networks. We also consider the TEASER method (Schäfer and Leser [2020]) from the early classification literature. We implement ForeClassNet and our experiments using Python (Van Rossum and Drake [2009]) and TensorFlow(Abadi et al. [2015]). To apply concrete dropout we use code from https://github.com/aurelio-amerio/ConcreteDropout. We implement MR Hydra,HC2, and TEASER using the AEON package (Middlehurst et al. [2024a]), and sktime (Löning et al. [2019]) for inception time. We train ForeClassNet using the filter set $\{3, 6, 12, 18, 24, 30, 36\}$ and the adaptive momentum optimization algorithm (Kingma and Ba [2015]) with a learning rate of 0.001. We train ForeClassNet and Inception time for 50 epochs in our third experiment and 100 epochs in the remaining experiments. We optimize them using a batch size of 64 in most experiments and a batch size of 128 in our second experiment. For HC2 we also specify a time contract and indicate how long the HC2 classifier takes to train in the results table of each of our experiments. We repeat each experiment five times with different tensor flow seeds from 0 to 4 which

impacts randomized tensor flow operations such as weight initialization. We then compute the mean and standard deviation across repetitions of the test set classification accuracy, test set F1 scores, and training time of each model in seconds, using the scikit-learn package(Pedregosa et al. [2011]) and the time package and rounding to 3 decimal places. In our experiments we set 80% of the data to be in the training set, 20% to be in the test set, and 10% of the training set to be a validation set. To assess the statistical significance of our results we use McNemar's test (McNemar [1947]) when comparing ForeClassNet with the best performing competing model in terms of test set classification accuracy in each experiment replication. We implement McNemar's test using the statsmodels package (Seabold and Perktold [2010]). The null hypothesis is the error probabilities of both classifiers being compared are equal, and the alternative hypothesis is that they are not equal. All our experiments are carried out locally on an Apple M1 Macbook pro (8-core CPU, 16 GB RAM) and 500.28 GB SSD. More information on the data used in the first experiment can be found in the technical appendix.

Table 1: Should a player adjust their position when trying to catch a ball?

Model	accuracy	Macro F1	Weighted F1	training time
ForeClassNet	0.772± 0.003	0.772± 0.003	0.772±0.003	438.666± 20.871
TEASER	0.626±0.008	0.625± 0.009	0.625± 0.008	16.088± 1.353
Inception	0.739± 0.003	0.738± 0.003	0.738±0.003	2207.026±57.297
MR-Hydra	0.688± 0.009	0.688± 0.009	0.688±0.009	95.086±8.739
HC2	0.742± 0.003	0.741± 0.003	0.741± 0.003	9831.522± 177.730

In Table 1 we display the results of a simulation of someone trying to catch a ball. We do this by simulating 2-D projectile motion, a time series of (x, y) coordinates, affected by wind, drag, and observation error. We produce 10,000 trajectories. The first 10 time points of the time series represent the motion of the ball in the first 0.25 seconds, at which point, the catcher must determine if they need to adjust their position. To do this they need to forecast the trajectory over the next 60 time points. The trajectory of a class is 1 if the player is going to need to adjust their position in some way, and 0 otherwise, with classes being approximately balanced. Due to the shorter length of the observed time series, we use the filter length set $\{3, 6, 9\}$ in ForeClassNet. ForeClassNet naturally extends to the multivariate setting, as it is a deep model, with the only changes being that we replicate the hand-crafted filters twice to apply to each dimension of the time series, and increase the nodes in the time-distributed dense layer from 1 to 2. We can see that ForeClassNet achieves superior test set accuracy compared to competing methods, with the resulting p-values of McNemar's test comparing ForeClassNet with the best competing method in terms of test set accuracy in each replication being [4.39e-05,2.33e-05,0.000909,5.93e-05,0.000813] to 3 significant figures, which are highly significant. Therefore, the error probabilities are statistically significantly different from each other, with ForeClassNet achieving superior test set accuracy in each replication. In addition to its classification output, ForeClassNet forecasts the ball's trajectory which can be used for precise positioning of the player, achieving a root mean squared error (RMSE) of 2.042 ± 0.064 which is a $79.24\% \pm 0.65\ \%$ improvement over the constant velocity extrapolator and a $78.32\% \pm 0.694\%$ improvement over a ballistic forecast which includes gravity but ignores wind and drag. We also train ForeClassNet where we only feed the model's estimates of the means and variances for the forecasted trajectory coordinates into the classification component, that is only the $10 + 1$ to the $10 + 60$ terms in (7). This gives a test set classification accuracy of 0.749 ± 0.021 and macro and weighted F1 scores of 0.745 ± 0.026 and 0.746 ± 0.026 respectively. This shows that in terms of the decision making framework of ForeClassNet the future observations contain a lot of information about the future class labels, with the observed values of the time series providing additional context to the classification component, slightly boosting test set classification accuracy by 0.023 ± 0.021. Overall, ForeClassNet achieves superior test set performance compared to the competing methods, and provides additional information in the form of the forecasted trajectory of the ball which could be used for precise positioning of the player, which would be helpful particularly if the predicted class label is 1.

In Table 2 we display our results from a challenging stock forecasting and classification problem using data from https://www.kaggle.com/datasets/tsaustin/us-historical-stock-prices-with-earnings-data/data, which contains historical adjusted closing prices of 1000s of stocks and the release dates of quarterly earnings reports (QERs). We have 96,614 time series from 2009 to 2021, with financial time series exhibiting a low signal to noise ratio. For a given stock we take the 40 adjusted closing

Table 2: Stock price data

Model	accuracy	Macro F1	Weighted F1	training time
ForeClassNet	0.425 ± 0.002	0.421 ± 0.004	0.421 ± 0.004	2136.408±525.117
Inception	0.430 ± 0.003	0.418 ± 0.007	0.418 ± 0.008	41999.07 ± 11657.28

prices prior to the release of its QER as the 'observed' time series, and the adjusted closing price immediately after the company's QER release to be forecasted. If the percentage change in price after the release of the company's QER is greater than or equal to 5% it is in class 1, if the percentage change is less than or equal to -5% it is in class 2, otherwise it is in class 0. This results in an unbalanced problem with class 0 being the vast majority, so we under sample class 0 leaving a data set of 17802 time series in class 0 and class 1, and 17634 in class 2. We then convert the price time series into raw return time series. We only train ForeClassNet and inception time as the current implementations of the other methods struggle to handle the large training set size, with smaller training set sizes resulting in worse test set generalization. We train both models with early stopping using a patience of 10 epochs. Due to the implementation of Inception time we fit several Inception time networks for an increasing number of epochs and choose the model with the largest validation set classification accuracy. ForeClassNet and Inception time achieve comparable test set accuracy with the resulting p-values from McNemar's test being [0.0353,0.621,0.984,0.506,0.0689] to 3 significant figures,which, excluding the first replication, indicate there is not a statistically significant difference in the error probabilities of the classifiers. ForeClassNet also outputs forecasts in addition to classification decisions which gives it an advantage over Inception time in this setting. The forecasting error of ForeClassNet as measured by RMSE is 0.124 ± 0.001, which is an improvement of $41.828\% \pm 0.437\%$ compared to the persistence forecast that has an RMSE of 0.213. Most financial forecasting is carried out in the mean squared error sense which can provide reasonable forecasts in terms of the magnitude of price changes, but can fail in providing the correct direction. However, ForeClassNet helps to solve this problem by providing both a forecast and a predicted direction through the classification output. We also train a ForeClassNet model where we only feed the forecast means and variance into the classification component rather than all the components of (7), which interestingly achieves a far lower test set accuracy of 0.333 ± 0.002 when compared to the results of the standard ForeClassNet in Table 2 unlike the results from our first experiment. The forecasts are not sufficient for predicting price movements but the multi-output approach of ForeClassNet helps to alleviate these problems along with the learned representations of the observed return series that ForeClassNet provides. Overall ForeClassNet achieves equivalent test set performance compared to the competing method, and provides additional information by giving both predictions on the magnitude and direction of price changes, allowing it to be a more complete decision making system beyond just a forecasting model or classification model.

Table 3: Distinguishing between two different autoregressive processes

Model	accuracy	Macro F1	Weighted F1	training time
ForeClassNet	0.996 ± 0.001	0.996 ± 0.001	0.996 ± 0.001	805.136 ± 15.913
TEASER	0.738 ± 0	0.738 ± 0	0.738 ± 0	256.564 ± 1.771
Inception	0.971 ± 0.001	0.971 ± 0.001	0.971 ± 0.001	2108.46 ± 74.003
MR-Hydra	0.790 ± 0.005	0.790 ± 0.005	0.790 ± 0.005	108.25 ± 11.553
HC2	0.878 ± 0.007	0.878 ± 0.008	0.878 ± 0.008	9665.014 ± 241.588

In Table 3 we show the results of a simulation in which we want to distinguish between time series $Y_t = 0.6Y_{t-1} - 0.3Y_{t-2} + 0.2Y_{t-3} + \epsilon_t$ with $\epsilon_t \sim N(0,1)$ and $Z_t = 0.7Z_{t-1} - 0.4Z_{t-2} + \eta_t$ with $\eta_t \sim N(0,1)$. We simulate 10,000 time series of length 50, with an approximate equal split between both classes. We set the first 40 realizations as observed and the last 10 realizations as unobserved future values. The resulting p-values for McNemar's test when comparing ForeClassNet with the inception time classifier are [3.91e-12,1.60e-10,2.59e-11,3.00e-11,3.42e-12] to 3 significant figures respectively, which are highly significant. Therefore, the error probabilities are significantly different from each other, with ForeClassNet achieving superior test set accuracy to all competing methods. We also investigate ForeClassNet's ability to distinguish between different noise generating processes, $Y_t = 0.6Y_{t-1} - 0.3Y_{t-2} + 0.2Y_{t-3} + \epsilon_t, \epsilon_t \sim N(0,1)$ and $Z_t = 0.6Z_{t-1} - 0.3Z_{t-2} +$

$0.2Z_{t-3} + \eta_t, \eta_t = exp\{\frac{h_t}{2}\}\xi_t, h_t = -1 + 0.85(h_{t-1} + 1) + 0.2\zeta_t, \xi_t \sim N(0,1), \zeta_t \sim N(0,1)$.We simulate 10,000 time series of length 50, with an approximate equal split between both classes. We set the first 40 realizations as observed and the last 10 realizations as unobserved future values. ForeClassNet achieves classification accuracy of 0.948 ± 0.005 and macro and weighted F1 scores of 0.947 ± 0.005, which was superior to TEASER and equivalent to the other state of the art methods.

Table 4: Classifying ECG data

Model	accuracy	Macro F1	Weighted F1	training time
ForeClassNet	0.960 ±0.004	0.634 ± 0.022	0.954± 0.005	2034.718± 59.921
TEASER	0.956± 0	0.609± 0	0.948±0	392.560±9.123
Inception	0.956± 0.003	0.666± 0.028	0.953±0.003	4520.538±34.830
MR-Hydra	0.960± 0.001	0.652± 0.029	0.954±0.002	52.444±2.852
HC2	0.962±0	0.642± 0.002	0.956±0	3692.86± 52.297

In Table 4 we display our results on the ECG 5000 TSC problem where the data can be found at: https://timeseriesclassification.com/description.php?Dataset=ECG5000. ECG classification is a challenging problem which is seeing increased research (e.g. Ismail et al. [2023]) due to the increased demand for remote healthcare. There are 5000 time series, each of length 140, and five classes representing different heart beats: Normal, R-on-T premature ventricular contraction, Supraventricular Premature or Ectopic beat, Premature ventricular contraction, and Unclassifiable Beat. This is an unbalanced problem with 2919 time series in class 1, 1767 in class 2, 194 in class 3, 96 in class 4, and 24 in class 5. We set the first 135 realizations as observed values and the last five realizations as unobserved future values. We can see ForeClassNet achieves comparable performance to the state of the art TSC methods. This is especially pleasing given the very small data set for training a deep neural network, with the p-values from using McNemar's test to compare ForeClassNet with HC2, the competing model with the largest test set accuracy in each replication, being [0.773,0.0389,0.453,0.814,0.803] to 3 significant figures. These are insignificant, excluding the second replication, which indicates that we fail to reject the null hypothesis; the error probabilities of ForeClassNet and HC2 are not significantly different from each other. Overall ForeClassNet achieves equivalent test set performance to the competing methods.

8 Conclusion

In this paper we propose and mathematically formalize a new temporal machine learning problem which we call 'Foreclassing', where we wish to make a classification decision based on past information, but the true class label is only revealed at a future time point. We prove the motivating Foreclassing Theorem which demonstrates why we should solve this problem. We propose a deep neural network which we call 'ForeClassNet' to solve this problem. To construct ForeClassNet we propose two novel neural network layers, Boltzmann Convolutions and Welford mean-variance layers. Through several experiments we demonstrate ForeClassNet achieves results which are comparable or superior to the existing state of the art time series classification models in terms of test set classification accuracy. Future research could explore new models for solving the Foreclassing problem or developing further theory to build upon the Foreclassing Theorem. There are also opportunities for further applications of Boltzmann convolutions and Welford mean-variance layers, such as integrating them into existing neural networks or creating new networks to solve problems of interest. Future work could also explore hypothesis tests to determine if future information is helpful and relevant for predicting the future class label. Further work could also be undertaken in terms of exploring applications of Foreclassing in neuroscience or robotics.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg,

Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Tessel Blom, Daniel Feuerriegel, Philippa Johnson, Stefan Bode, and Hinze Hogendoorn. Predictions drive neural representations of visual events ahead of incoming sensory information. *Proceedings of the National Academy of Sciences*, 117(13):7510–7515, 2020.

Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic Convolution: Attention Over Convolution Kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11030–11039, 2020.

Andreas Damianou and Neil D Lawrence. Deep Gaussian Processes. In *Artificial Intelligence and Statistics*, pages 207–215. PMLR, 2013.

Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. Hydra: Competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery*, 37(5): 1779–1805, 2023.

Philip Doganis, Eleni Aggelogiannaki, and Haralambos Sarimveis. A combined model predictive control and time series forecasting framework for production-inventory systems. *International Journal of Production Research*, 46(24):6841–6853, 2008.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *International Conference on Machine Learning*, pages 1050–1059. PMLR, 2016.

Yarin Gal, Jiri Hron, and Alex Kendall. Concrete Dropout. *Advances in Neural Information Processing Systems*, 30, 2017.

Shizhong Han, Zibo Meng, Zhiyuan Li, James O'Reilly, Jie Cai, Xiaofeng Wang, and Yan Tong. Optimizing Filter Size in Convolutional Neural Networks for Facial Action Unit Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5070–5078, 2018.

Hinze Hogendoorn and Anthony N Burkitt. Predictive coding of visual object position ahead of moving objects revealed by time-resolved EEG decoding. *NeuroImage*, 171:55–61, 2018.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

Ali Rida Ismail, Slavisa Jovanovic, Naeem Ramzan, and Hassan Rabah. ECG Classification Using an Optimal Temporal Convolutional Network for Remote Health Monitoring. *Sensors*, 23(3):1697, 2023.

Ali Ismail-Fawaz, Maxime Devanne, Jonathan Weber, and Germain Forestier. Deep Learning For Time Series Classification Using New Hand-Crafted Convolution Filters. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 972–981. IEEE, 2022.

Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F Schmidt, Jonathan Weber, Geoffrey I Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inception Time: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

David C Knill and Alexandre Pouget. The bayesian brain: the role of uncertainty in neural coding and computation. *TRENDS in Neurosciences*, 27(12):712–719, 2004.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation Applied to Handwritten Zip code Recognition. *Neural Computation*, 1(4):541–551, 1989.

Ian Lenz, Ross A Knepper, and Ashutosh Saxena. DeepMPC: Learning Deep Latent Features for Model Predictive Control. In *Robotics: Science and Systems*, volume 10, page 25. Rome, Italy, 2015.

Shen Liu, Elizabeth Ann Maharaj, and Brett Inder. Polarization of forecast densities: A new approach to time series classification. *Computational Statistics & Data Analysis*, 70:345–361, 2014.

Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, and Franz J Király. sktime: A unified interface for machine learning with time series. *Advances in Neural Information Processing Systems*, 33, 2019.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Matthew Middlehurst, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall. HIVE-COTE 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110(11):3211–3243, 2021.

Matthew Middlehurst, Ali Ismail-Fawaz, Antoine Guillaume, Christopher Holder, David Guijo-Rubio, Guzal Bulatova, Leonidas Tsaprounis, Lukasz Mentel, Martin Walter, Patrick Schäfer, et al. AEON: a python toolkit for learning from time series. *Journal of Machine Learning Research*, 25(289): 1–10, 2024a.

Matthew Middlehurst, Patrick Schäfer, and Anthony Bagnall. Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, pages 1–74, 2024b.

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

Patrick Schäfer and Ulf Leser. Teaser: early and accurate time series classification. *Data Mining and Knowledge Discovery*, 34(5):1336–1362, 2020.

Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, page 125, 2016.

Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

Gautham Vasan, Mohamed Elsayed, Seyed Alireza Azimi, Jiamin He, Fahim Shahriar, Colin Bellinger, Martha White, and Rupam Mahmood. Deep Policy Gradient Methods Without Batch Updates, Target Networks, or Replay Buffers. *Advances in Neural Information Processing Systems*, 37: 845–891, 2024.

Tian Wang, Zhaoying Liu, Ting Zhang, and Yujian Li. Time Series Classification Based on Multiscale Dynamic Convolutional Features and Distance Features. In *2021 2nd Asia Symposium on Signal Processing (ASSP)*, pages 239–246. IEEE, 2021.

Barry Payne Welford. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics*, 4(3):419–420, 1962.

Yazhan Zhang, Weihao Yuan, Zicheng Kan, and Michael Yu Wang. Towards Learning to Detect and Predict Contact Events on Vision-based Tactile Sensors. In *Conference on Robot Learning*, pages 1395–1404. PMLR, 2020.

Technical Appendix

Should a player adjust their position when trying to catch a ball?

In Table 1 of Section 7 we display the results from a physics simulation of someone catching a ball. We generate 10,000 2D projectile trajectories simulating a ball thrown towards a player. Each trajectory consists of 10 observed time points representing the part of the ball's trajectory the player has seen, and 60 future time points which represent where the ball will be in the future. The player then needs to decide based only on what they have seen if they need to adjust their position or not to catch the ball successfully. We discretize time where each time step represents 0.025 seconds. For each trajectory, the initial launch velocity v_0 is sampled uniformly from [10,15] m/s, the launch angle θ is sampled uniformly from [20,40] degrees which is then converted to radians. We sample the initial height of the ball y_0 uniformly from [1,1.1]m and set the horizontal starting point x_0 to be 0m. From this we then compute the velocity components $v_x = v_0 cos\theta$ and $v_y = v_0 sin\theta$. We simulate per trajectory a single baseline level of wind according to $N(0, 1)m/s$ and add an additional single level of wind during the future time points from $N(0, 6)$ m/s. We simulate drag uniformly from 0.03 to 0.05 during the observed time points and uniformly from 0.08 to 0.12 during the future time points. We compute the horizontal acceleration as $a_x = wind - drag \times v_x$ and the vertical acceleration as $a_y = -9.81 - drag \times v_y$. We then update the velocities and subsequently the x and y coordinates of the projectile using Euler's method. We further add noise to the x and y coordinates at each observed and future time step simulated independently from $N(0, 0.15)$. We assume the player is operating along the X-axis 12-14m away from the source of the ball, the catch zone. Within the catch zone, for a given trajectory, we pick the highest point the 'ball reaches. If this is below 1m we say that the player can stay in their current position, otherwise the player needs to adjust their position.