Enhancing Time Series Forecasting with Changepoint-Aware Training: An ALPIN-Enhanced DeepAR Approach

Your Name

Your Institution

Abstract

Time series forecasting in real-world applications often encounters non-stationary data characterized by structural breaks or changepoints, where the underlying statistical properties shift abruptly. Training probabilistic forecasting models such as DeepAR across these regime changes can degrade performance by forcing the model to learn patterns that span fundamentally different dynamics. We present a novel approach that integrates changepoint detection with deep learning-based forecasting through a method we call BatchCP (Batch Changepoint) filtering. Our approach leverages ALPIN (Adaptive Learning of Penalty for INference), a supervised learning algorithm that automatically determines optimal penalty parameters for changepoint detection, to identify structural breaks in training data. We then filter training batches whose context windows overlap with detected changepoints, ensuring the forecasting model trains only on homogeneous temporal segments. This paper presents the theoretical foundation, algorithmic implementation, and experimental methodology for this changepoint-aware training paradigm. By combining rigorous changepoint detection with modern probabilistic forecasting, we propose a principled framework for handling non-stationary time series in data-driven forecasting applications.

Keywords: changepoint detection, time series forecasting, deep learning, ALPIN, DeepAR, BatchCP filtering, non-stationary data

Enhancing Time Series Forecasting with Changepoint-Aware Training: An ALPIN-Enhanced DeepAR Approach

Introduction

Time series forecasting plays a critical role in numerous domains, including finance, energy management, supply chain optimization, and healthcare monitoring. Modern deep learning approaches, particularly probabilistic methods such as DeepAR salinas2020deepar, have demonstrated remarkable success in capturing complex temporal dependencies and providing uncertainty estimates. However, a fundamental challenge remains largely unaddressed: real-world time series are frequently non-stationary, exhibiting abrupt changes in statistical properties known as changepoints or structural breaks truong2020selective.

The presence of changepoints poses a significant challenge for neural forecasting models. Standard training procedures use sliding windows to extract numerous batches from historical data, with each batch consisting of an encoder window (context) and a decoder window (forecast target). When these windows span across changepoints, the model must simultaneously learn patterns from fundamentally different regimes. This "contamination" of training batches can lead to suboptimal gradient updates, confused internal states, and ultimately degraded forecast accuracy.

Traditional approaches to handling non-stationarity in forecasting typically involve either assuming stationarity after preprocessing (e.g., differencing, detrending) or incorporating explicit regime-switching mechanisms into the model architecture. However, these approaches either fail to capture abrupt structural breaks or require knowing changepoint locations a priori. To our knowledge, no existing work systematically addresses the contaminated batch problem by explicitly filtering training data based on detected changepoints.

This paper introduces a novel changepoint-aware training methodology that bridges changepoint detection and probabilistic forecasting. Our approach consists of two main

components: (1) automatic changepoint detection using ALPIN truong2017automatic, a supervised learning algorithm that optimally tunes penalty parameters for changepoint identification, and (2) BatchCP filtering, which removes training batches whose encoder windows overlap with detected changepoints. This ensures that the forecasting model learns only from temporally consistent data segments.

Research Questions and Contributions

This work addresses the following research questions:

1. How can supervised changepoint detection be effectively integrated into deep learning-based forecasting pipelines?
2. Does filtering training batches based on detected changepoints improve forecast accuracy compared to standard training?
3. What is the appropriate trade-off between training data quantity (reduced by filtering) and quality (improved by removing contaminated batches)?

Our primary contributions are:

1. A complete implementation of the ALPIN algorithm for automatic penalty parameter learning in changepoint detection, including risk minimization, optimal partition solving, and comprehensive evaluation metrics.
2. The BatchCP filtering methodology, which systematically removes contaminated training batches from the forecasting pipeline based on detected changepoint locations.
3. An experimental framework comparing baseline DeepAR forecasting with ALPIN-enhanced DeepAR on piecewise-constant time series, providing insights into the benefits of changepoint-aware training.

4. Open-source Python implementation integrating ALPIN with PyTorch-based forecasting, facilitating reproducibility and extension to other forecasting architectures.

Paper Organization

The remainder of this paper is organized as follows. Section 2 provides background on changepoint detection, the ALPIN algorithm, and DeepAR forecasting. Section 3 details our methodology, including the ALPIN implementation, BatchCP filtering procedure, and experimental design. Section 4 will present empirical results (to be completed), and Section 5 will discuss implications and future directions.

Background and Related Work

Changepoint Detection

Changepoint detection addresses the problem of identifying locations in a time series where statistical properties change abruptly. Formally, given a signal $y = (y_1, \ldots, y_T)$, a changepoint at time τ divides the signal into two segments with different distributional characteristics. The changepoint detection problem seeks to identify all such locations $\{\tau_1, \ldots, \tau_K\}$ in the signal.

The optimal partitioning approach frames changepoint detection as an optimization problem. Given a signal y and a partition A dividing it into segments, we define the penalized risk:

$$R_\beta(y, A) = R(y, A) + \beta|A| \tag{1}$$

where $R(y, A)$ is the empirical risk (typically sum of squared residuals for piecewise-constant signals), $|A|$ is the number of segments, and $\beta > 0$ is a penalty parameter controlling the trade-off between model fit and complexity. The optimal partition is then:

$$\hat{A}(\beta) = \arg\min_{A} R_\beta(y, A) \tag{2}$$

Computing $\hat{A}(\beta)$ efficiently is non-trivial, as the search space grows exponentially with signal length. The PELT (Pruned Exact Linear Time) algorithm killick2012optimal solves this problem using dynamic programming with an optimal worst-case complexity of $O(T)$ for favorable data and $O(T^2)$ in general. PELT exploits the property that candidate changepoints can be pruned when they cannot possibly improve the optimal solution.

A comprehensive review by truong2020selective categorizes changepoint detection methods into three families: kernel-based methods, optimal partitioning methods (including PELT), and binary segmentation approaches. Optimal partitioning methods are preferred when the exact number and locations of changepoints must be identified, while binary segmentation provides faster approximate solutions.

The critical challenge in optimal partitioning is selecting an appropriate value for the penalty parameter β. Traditional approaches include cross-validation, information criteria (AIC, BIC), and expert tuning. However, these methods are computationally expensive, data-dependent, and often fail to generalize across different signal characteristics.

ALPIN: Adaptive Learning of Penalty for INference

ALPIN truong2017automatic addresses the penalty parameter selection problem through supervised learning. Given a training dataset of signals $\{y_i\}_{i=1}^N$ with ground truth changepoint annotations $\{A_i^{lab}\}_{i=1}^N$, ALPIN learns the optimal β that minimizes the average excess risk across the training set.

The excess risk $E(y, \beta)$ for a signal y with labeled partition A^{lab} is defined as:

$$E(y, \beta) = R(y, \hat{A}(\beta)) - R(y, A^{lab}) \tag{3}$$

This measures the difference in empirical risk between the automatically detected partition $\hat{A}(\beta)$ and the ground truth A^{lab}. The excess risk is always non-negative: it equals

zero when the detected partition matches the labeled partition perfectly and increases as they diverge.

ALPIN formulates the learning problem as:

$$\beta^* = \arg\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} E(y_i, \beta) \tag{4}$$

The optimization employs a two-phase strategy:

1. **Warm Start**: A single signal is randomly selected from the training set, and β is optimized on this signal alone. This provides a good initialization for the global optimization phase.

2. **Global Optimization**: Using the warm start value as initialization, L-BFGS-B optimization minimizes the average excess risk across all training signals. The optimization is performed in the log-domain ($\log \beta$) to ensure positivity and handle the wide range of possible values.

At each iteration of the optimization, computing $E(y_i, \beta)$ requires solving the optimal partition problem via PELT for both the candidate β and the labeled partition. This makes ALPIN computationally intensive, with complexity $O(N \cdot T^2)$ per optimization iteration, but the resulting β^* generalizes well to new signals from the same domain.

A key advantage of ALPIN is its adaptability to different annotation protocols. For instance, Protocol I may label all changepoints regardless of magnitude, while Protocol II may label only large-amplitude changes (e.g., $|\Delta\mu| > 3\sigma$). ALPIN automatically learns different optimal β values for each protocol, directly encoding the annotator's implicit criteria into the penalty parameter.

Time Series Forecasting with DeepAR

DeepAR salinas2020deepar is a probabilistic forecasting method based on autoregressive recurrent neural networks. Unlike traditional ARIMA models or

point-estimate neural approaches, DeepAR produces full predictive distributions, enabling uncertainty quantification crucial for decision-making under risk.

The model architecture consists of an encoder-decoder LSTM network. During training, the encoder processes a context window of length T_{enc} to build a hidden state representation, which the decoder then uses to autoregressively predict a forecast window of length T_{dec}. The model is trained to maximize the likelihood of the observed data under a parametric distribution (e.g., Gaussian, negative binomial) whose parameters are predicted by the network.

A distinguishing feature of DeepAR is its ability to learn across multiple related time series simultaneously. By sharing parameters across series while allowing for series-specific effects (through learned embeddings), DeepAR can leverage cross-series information to improve forecasts, particularly for cold-start scenarios with limited historical data.

Training DeepAR involves creating numerous training batches from sliding windows over the available history. For a single time series of length T, one can extract up to $T - T_{enc} - T_{dec} + 1$ training examples. This data augmentation is essential for training deep networks but introduces a critical issue: many batches will inevitably span across changepoints in the underlying data-generating process.

When a training batch's encoder window contains a changepoint, the LSTM must compress information from two different regimes into a single hidden state. This forces the network to learn a compromise representation that poorly captures either regime's dynamics. Similarly, when the changepoint occurs within the decoder window, the autoregressive predictions must transition between regimes mid-forecast, a task that is fundamentally at odds with the smoothness assumptions implicit in the RNN architecture.

Related Work on Non-Stationary Forecasting

Several approaches have been proposed to handle non-stationary time series in deep learning-based forecasting. wen2017multi incorporate hierarchical structure and multi-task learning to share information across related series with different dynamics. However, their

approach does not explicitly account for structural breaks within individual series.

Regime-switching models provide an alternative paradigm by explicitly modeling transitions between different dynamical states. Hidden Markov Models (HMMs) and their variants have long been used for this purpose in classical time series analysis. More recently, deep learning architectures have incorporated switching mechanisms, such as rangapuram2018deep who propose a state-space model with learned switching dynamics. These methods, however, require the model to simultaneously learn the regime structure and the within-regime dynamics, increasing model complexity and data requirements.

Online changepoint detection methods adams2007bayesian update changepoint beliefs as new data arrives, enabling adaptive forecasting. However, these methods are designed for inference at prediction time rather than improving training data quality, which is our focus.

To our knowledge, no prior work has systematically applied offline changepoint detection to filter training data for deep learning-based forecasting. The BatchCP methodology we propose fills this gap by using ALPIN-detected changepoints to curate training batches, ensuring the model learns only from temporally consistent data.

Methodology

Problem Formulation

We consider the following forecasting problem: given a collection of time series $\{y^{(s)}\}_{s=1}^{S}$, where each series $y^{(s)} = (y_1^{(s)}, \ldots, y_T^{(s)})$ contains potential changepoints, we aim to train a probabilistic forecasting model that produces accurate multi-step-ahead forecasts and well-calibrated predictive distributions.

Let $\mathcal{C}^{(s)} = \{\tau_1^{(s)}, \ldots, \tau_{K_s}^{(s)}\}$ denote the set of changepoint locations in series s, where $1 < \tau_k^{(s)} < T$ for all k. These changepoints partition the series into $K_s + 1$ homogeneous segments, where statistical properties (mean, variance, autocorrelation structure) remain constant within each segment but may differ across segments.

The standard training procedure for DeepAR creates batches by sampling starting

positions t uniformly from $[T_{enc}, T - T_{dec}]$ and extracting:

- Encoder window: $x_t = (y^{(s)}_{t-T_{enc}+1}, \ldots, y^{(s)}_t)$
- Decoder window: $y_t = (y^{(s)}_{t+1}, \ldots, y^{(s)}_{t+T_{dec}})$

A batch is *contaminated* if its encoder window $[t - T_{enc} + 1, t]$ contains one or more changepoints from $\mathcal{C}^{(s)}$. Our objective is to identify and remove such batches to create a *clean* training set.

ALPIN-Based Changepoint Detection

Implementation

Our implementation of ALPIN follows the algorithm proposed by truong2017automatic with several practical enhancements. The core ALPIN class provides a scikit-learn-style API with `fit()` and `predict()` methods.

The `fit(signals, ground_truths)` method implements the two-phase optimization:

Algorithm 1 ALPIN Training Algorithm

Require: Training signals $\{y_i\}_{i=1}^N$, labeled partitions $\{A_i^{lab}\}_{i=1}^N$

Require: Penalty bounds $[\beta_{min}, \beta_{max}]$

Phase 1: Warm Start

$i^* \leftarrow$ random selection from $\{1, \ldots, N\}$

$\beta_{warm} \leftarrow \arg\min_\beta E(y_{i^*}, \beta)$ using L-BFGS-B

Phase 2: Global Optimization

$\beta^* \leftarrow \arg\min_\beta \frac{1}{N} \sum_{i=1}^N E(y_i, \beta)$ starting from β_{warm}

return β^*

For each candidate β during optimization, we compute the excess risk by:

1. Solving the optimal partition problem $\hat{A}(\beta) = \arg\min_A R_\beta(y, A)$ using PELT via the `ruptures` library
2. Computing empirical risk on the detected partition: $R(y, \hat{A}(\beta))$
3. Computing empirical risk on the labeled partition: $R(y, A^{lab})$
4. Calculating excess risk: $E(y, \beta) = R(y, \hat{A}(\beta)) - R(y, A^{lab})$

Optimization is performed in the log-domain to ensure $\beta > 0$ and to improve numerical stability across the wide range of possible penalty values. The L-BFGS-B optimizer from `scipy.optimize` is used with default convergence criteria.

Evaluation Metrics

To assess the quality of detected changepoints, we implement several metrics from the changepoint detection literature:

Precision and Recall: Using a tolerance margin m (typically 5-10 samples), we define:

$$\text{Precision} = \frac{|\{cp \in \hat{\mathcal{C}} : \exists cp' \in \mathcal{C}^{lab}, |cp - cp'| \leq m\}|}{|\hat{\mathcal{C}}|} \tag{5}$$

$$\text{Recall} = \frac{|\{cp \in \mathcal{C}^{lab} : \exists cp' \in \hat{\mathcal{C}}, |cp - cp'| \leq m\}|}{|\mathcal{C}^{lab}|} \tag{6}$$

Hausdorff Distance: Measures the maximum distance between detected and true changepoints:

$$d_H(\hat{\mathcal{C}}, \mathcal{C}^{lab}) = \max\left\{\max_{cp \in \hat{\mathcal{C}}} \min_{cp' \in \mathcal{C}^{lab}} |cp - cp'|, \max_{cp' \in \mathcal{C}^{lab}} \min_{cp \in \hat{\mathcal{C}}} |cp - cp'|\right\} \tag{7}$$

Rand Index: Treats changepoint detection as a clustering problem on consecutive time points and measures agreement between detected and labeled segmentations.

BatchCP Filtering

Filtering Criterion

Given detected changepoints $\mathcal{C}^{(s)}$ for each training series s, we implement a `ChangePointAwareDataLoader` wrapper around PyTorch's standard `DataLoader`. For each

batch sampled by the underlying loader, we check whether its encoder window overlaps with any changepoint.

Formally, a batch starting at position t in series s is retained if and only if:

$$\forall \tau \in \mathcal{C}^{(s)} : (\tau < t - T_{enc} - \delta) \vee (\tau > t + \delta) \tag{8}$$

where δ is a tolerance parameter (typically 2-5 samples) providing a safety margin around changepoints. This margin accounts for:

1. Imperfect changepoint detection (tolerance in matching ground truth)
2. Transition periods where dynamics gradually shift rather than changing instantaneously
3. Edge effects in the LSTM's processing of the encoder window

Implementation

The `ChangePointAwareDataLoader` class wraps an existing PyTorch `DataLoader` and filters batches on-the-fly during iteration:

```
class ChangePointAwareDataLoader:
    def __init__(self, dataloader, changepoints_dict,
                 encoder_length, tolerance=2):
        self.dataloader = dataloader
        self.changepoints_dict = changepoints_dict
        self.encoder_length = encoder_length
        self.tolerance = tolerance


    def _batch_contains_changepoint(self, batch):
        x_dict, y = batch
        encoder_time_idx = x_dict["encoder_time_idx"]
```

```
        groups = x_dict["groups"]

        for i in range(batch_size):
            t_start = encoder_time_idx[i].min().item()
            t_end = encoder_time_idx[i].max().item()
            series_id = f"series_{groups[i, 0].item()}"

            for cp in self.changepoints_dict[series_id]:
                if (t_start - self.tolerance) <= cp <=
                   (t_end + self.tolerance):
                    return True
        return False

    def __iter__(self):
        for batch in self.dataloader:
            if not self._batch_contains_changepoint(batch):
                yield batch
```

This implementation preserves the standard PyTorch training loop while transparently filtering contaminated batches. The `ChangePointAwareDataLoader` maintains statistics on the number of batches processed and filtered, providing transparency into the data curation process.

Experimental Design

Data Generation

We generate synthetic piecewise-constant time series for controlled experiments. Each signal consists of multiple segments with constant mean values and additive Gaussian noise. Changepoints occur at random locations, with the number of segments and segment means sampled from specified distributions.

Formally, for a signal of length T with K changepoints at positions $\{\tau_k\}_{k=1}^{K}$:

$$y_t = \mu_k + \epsilon_t, \quad \tau_{k-1} < t \leq \tau_k \tag{9}$$

where $\mu_k \sim \mathcal{U}[-5, 5]$ are segment means and $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise with standard deviation σ.

The synthetic data generation module (`alpin.data.synthetic`) provides flexible control over:

- Number of signals and signal length
- Noise level σ
- Annotation protocol (Protocol I: all changepoints, Protocol II: only large jumps $|\mu_k - \mu_{k-1}| > 3\sigma$)
- Minimum segment length (to avoid trivially short segments)

DeepAR Configuration

We use the DeepAR implementation from the `pytorch-forecasting` library with the following configuration:

- **Encoder length** $T_{enc} = 60$: Context window of 60 time steps
- **Prediction length** $T_{dec} = 20$: Forecast 20 steps ahead
- **Hidden size**: 32 units per LSTM layer
- **Number of layers**: 2 stacked LSTM layers
- **Dropout**: 0.1 between LSTM layers
- **Distribution**: Gaussian with predicted mean and variance
- **Optimizer**: Adam with learning rate 10^{-3}

- **Batch size**: 32
- **Training epochs**: 20

Baseline vs. ALPIN-Enhanced Training

We compare two training procedures:

Baseline DeepAR: Standard training using all available batches from the training set. The PyTorch Lightning `Trainer` iterates over the full dataset for the specified number of epochs.

ALPIN-Enhanced DeepAR: Training with BatchCP filtering applied. The procedure is:

1. Train ALPIN on a labeled subset (or the full training set with synthetic labels)
2. Apply ALPIN to detect changepoints in all training series: $\mathcal{C}^{(s)} = \text{ALPIN.predict}(y^{(s)})$
3. Construct `ChangePointAwareDataLoader` with detected changepoints
4. Train DeepAR using the filtered dataloader

Both models use identical architectures, hyperparameters, and validation sets. The only difference is the training data: baseline uses all batches, while ALPIN-enhanced uses only batches passing the BatchCP filter.

Evaluation Framework

Model performance is evaluated on a held-out test set using standard forecasting metrics:

- **Mean Absolute Error (MAE)**: Average absolute difference between predictions and actuals
- **Root Mean Squared Error (RMSE)**: Square root of average squared errors, emphasizing larger deviations

- **Quantile Loss**: For probabilistic forecasts at various quantiles (10%, 50%, 90%)
- **Coverage**: Proportion of actuals falling within predicted confidence intervals

Metrics are computed across all test series and forecast horizons. Statistical significance of performance differences is assessed using paired tests (e.g., Wilcoxon signed-rank test) across per-series error distributions.

Ablation Studies

To understand the impact of key hyperparameters, we plan ablation studies varying:

- Tolerance parameter δ in BatchCP filtering (0, 2, 5, 10 samples)
- Encoder length T_{enc} (30, 60, 120 steps)
- Training set size and changepoint density
- Noise level σ affecting both ALPIN detection accuracy and forecasting difficulty

These experiments will provide insights into the robustness of the BatchCP methodology across different problem settings.

Results

[To be completed: This section will present experimental results comparing baseline and ALPIN-enhanced DeepAR across multiple metrics and problem settings.]

Discussion and Future Work

[To be completed: This section will interpret results, discuss limitations, and outline extensions to other forecasting architectures and real-world datasets.]

Conclusion

We have presented a novel methodology for integrating changepoint detection with deep learning-based time series forecasting. By implementing the ALPIN algorithm for automatic penalty parameter learning and introducing the BatchCP filtering technique, we

provide a principled approach to handling non-stationary data in neural forecasting pipelines. Our experimental framework enables systematic evaluation of changepoint-aware training, with results and analysis to be presented upon completion of experiments. This work opens new avenues for research at the intersection of changepoint analysis and deep learning, with potential applications across diverse forecasting domains.