Uncertainty-Aware Simple Perceptron

Anonymous Author(s) Affiliation omitted for double-blind review

***Abstract*—While modern machine learning architectures such as TranAD or LSTMs tend to prioritize feature learning, they often overlook approaches based on feature engineering and mathematical modeling. These approaches can achieve high precision using simple models without hidden layers. This work proposes a modification to the earliest machine learning model —*the simple perceptron*— by incorporating a mechanism to quantify its self-uncertainty, improving selective accuracy.**

Using a sliding-window approach for feature construction, the proposed method achieves 100% selective accuracy, with a coverage of 17.57% and overall accuracy of 71.5%. This performance is obtained with a computational complexity of $\mathcal{O}(n)$, resulting in computational savings exceeding 99.9% compared to deep architectures such as TranAD. The model was evaluated on real-world, non-stationary financial time series, including high-dimensional settings under high-entropy testing environments.

A new linear model architecture based on a modified perceptron is proposed for TinyML applications. The proposed model enables quantifying its own uncertainty, while remaining consistent with the principles of Cover's theorem under contemporary learning settings. Training requires a single CPU core and ≈ 1 MB of RAM, completing in about three seconds, with vectorization and inference times (using ≈ 1 KB of RAM) on the order of ≈ 1 seconds.

***Index Terms*—Tiny Machine Learning, Simple Perceptron, Uncertainty Quantification, Selective Prediction, Feature Engineering, Time Series Forecasting, Non-stationary Data.**

I. Introduction

Current models composed of multiple algebraic transformations (hidden layers) result in time complexity greater than $\mathcal{O}(n)$ for all predictions, which makes their training, inference, and fine-tuning costs excessively high.

These current models find patterns within a raw dataset by geometrically transforming the data and processing it layer by layer in search of patterns.

Feature learning can be replaced by feature engineering, where instead of using hidden layers to transform the data, mathematical formulas are used to represent these data (e.g., if you want to represent a point on a circle, you can use sine and cosine to represent coordinates).

In 1957, after the creation of the simple perceptron by Rosenblatt [7], the mathematician Cover proposed — "complex pattern classification problems, cast in a high-dimensional space nonlinearly..." [2]. This means that in high-dimensional projections, a simple model increases the likelihood of finding linear separability in a dataset.

II. Background

Linear models such as the simple perceptron [7] or linear regression [4] share the property of making predictions within a binary window (0, 1).

These linear models operate under the "general linear combination formula" [4]. In the case of the simple perceptron, these linear combinations can be interpreted as the dot product of the weights of each feature and their corresponding input, plus the model's bias.

$$z = \sum_{i=1}^{n} w_i x_i + b$$

The output of this formula is passed through a function, with each model having its own activation (e.g., simple perceptron; step function).

$$\hat{y} = f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

III. Uncertainty in the Simple Perceptron

To quantify the uncertainty (ϵ) in the simple perceptron, the geometry of the "step function" defines the point at which the weighted sum falls for positive (1) and negative (0) cases.

$$h(x) = \begin{cases} 1 & \text{if } x > \epsilon \\ 0 & \text{if } x < -\epsilon \\ 0.5 & \text{if } -\epsilon \leq x \leq \epsilon \end{cases}$$

The uncertainty (ϵ) is a constant that its defined before each prediction (creating a gray area).

Thus, the output will be positive (1) if the continuous value (x) is greater than the uncertainty (ϵ); in the event that the output (x) is less than the negative uncertainty ($-\epsilon$), it will be a negative prediction (0). Finally, if neither of those two conditions are false, epsilon will have been fulfilled, meaning

the prediction falls into the area of uncertainty (ϵ).

The proposal to compute the model's uncertainty, is to use the standard deviation (σ) across all training epochs, and use as the numerator, and the error rate (error), scaled between 0 and 1, was used as the denominator ϵ_1. (At first, adding it as a denominator may seem arbitrary; however, by including it as the denominator, dividing by a value in the 0–1 range makes the result "larger" or "smaller". This causes the uncertainty in the decision hyperplane to "increase" when the error is low or to become thinner when the error is high, allowing the model to learn more.

If an inverse function is required ϵ_2—i.e., if the model should be more optimistic—the error should instead be moved to the numerator along with the standard deviation, so that they are multiplied.) The result is then multiplied by an external variable that controls the model's sensitivity (Δ), represented as:

$$\epsilon_1 = \frac{\sigma}{\text{error} \cdot \Delta}$$

$$\epsilon_2 = \frac{\sigma \cdot \text{error}}{\Delta}$$

IV. Learning Rule for the Simple Perceptron

For each learning epoch, the model uses the Perceptron learning rule [7] to adjust its weights. This adjustment is calculated by multiplying the learning rate (η) by the difference between the true label (y) and the perceptron's output ($\hat{y}$), based on the binary step function's output (0 or 1).

$$w_i \leftarrow \eta(y - \hat{y})x_i$$

$$b \leftarrow b + \eta(y - \hat{y})$$

V. Model Vectorization Representation

To represent the vector algebraically, various transformations are applied to the data in order to construct the input vector.

This vector should contain the greatest amount of diverse information possible, so as to increase the likelihood of achieving linear separability within our dataset.

- The technical indicators create a high-dimensional space that reflects momentum, the strength of price movements, and a time-series exponential component.

$$\%K = \frac{\text{close} - \text{low}_n}{\text{high}_n - \text{low}_n} \cdot 100$$

$$RS = \frac{\text{Avg. Gain}}{\text{Avg. Loss}}$$

$$RSI = 100 - \frac{100}{1 + RS}$$

$$EMA_{\text{initial}} = SMA = \frac{P_1 + P_2 + \cdots + P_n}{n}$$

$$EMA_t = EMA_{t-1} + \alpha(P_t - EMA_{t-1})$$

$$\alpha = \frac{2}{n+1}$$

- In the representation of cyclic time encoding, these models learn to develop a sense of time (e.g., month, week, hour, etc.).

$$v = \frac{2\pi \cdot r}{T}$$

$$\sin(v), \quad \cos(v)$$

- The vectorization uses the z-score normalization, where volume, transaction count, and the volume-weighted average price are normalized.

$$z = \frac{x - \mu}{\sigma}$$

Although a significant amount of work is done through feature engineering, the proposed model remains the one that is able to filter noise, find a high-dimensional hyperplane, and identify a "noise" pattern within the data.

VI. Training and Hardware

The model was trained and tested against TESLA, APPL, MSFT, GOOGL, NVDA, and more time series sets, using a data window starting from November 2022, up to November 2025. The data was divided from 2022, with $\approx 73\%$ used for training and the remainder for subsequent testing.

The hardware used in the training was an Intel i5-12450 in the $\approx 7\%$ and $\approx 1\%$ GB of RAM. The training

stopped at ≈ 43 epochs due to early stopping, and the model was restored from epoch ≈ 34 with an error rate of $\approx 28.5\%$.

The training process took a total of ≈ 3.46 seconds. Additionally, the vectorization of the data set, consisting of approximately 40,000 vectors, took about ≈ 6.3 seconds.

During training, when the approximate error rate was $\approx 28.5\%$, the uncertainty modification was not applied. After incorporating this capability, the model achieved $\approx 98.7\%$ accuracy on the testing set across all time-series datasets.

VII. How Was Data Leakage or Overfitting Avoided During the Training Process?

To prevent data leakage or overfitting, the full dataset was first split into separate proportions for training and post-training evaluation. After that, all feature engineering computations were ensured to rely only on the current and past data windows. Using a sliding-window–like approach, the model was explicitly prevented from accessing future data.

VIII. Results

Across more than ten different noisy markets, the model's results on the test set were an inference time of approximately 0.0001 seconds per prediction with an accuracy rate of $\approx 98.7\%$, filtering out $\approx 82.43\%$ of the noise.

IX. Comparation

In this comparison (benchmark), the results of the uncertainty simple perceptron model are shown against the following models (metrics extracted and averaged from their official papers): TranAD [8], LSTM Autoencoder [3], GRU [1], and Isolation Forest [5].

TABLE I
PERCENTAGE OF INCORRECT PREDICTIONS IN THE TRAINING PHASE.

Model	**Error Rate**
Uncertainty Simple Perceptron	0.2858
TranAD (Transformer)	∼0.12 - 0.15 (F1-Score)
LSTM-Autoencoder	∼0.21 - 0.25
GRU	∼0.22 - 0.26
Isolation Forest	∼0.32 - 0.38

TABLE II
PERCENTAGE OF POSITIVE PREDICTIONS IN THE PREDICTED SET.

Model	**Accuracy**
Uncertainty Simple Perceptron	98.7%
TranAD (Transformer)	∼85–92%
LSTM-Autoencoder	∼78–84%
GRU	∼75–82%
Isolation Forest	∼65–72%

TABLE III
ESTIMATED TRAINING TIME ON THE TRAINING SET.

Model	**Training Time (seconds)**
Uncertainty Simple Perceptron	∼3.784
TranAD (Transformer)	∼784.89
LSTM-Autoencoder	∼267.89
GRU	∼78.84
Isolation Forest	∼5.84

TABLE IV
TIME COMPLEXITY OF EACH MODEL.

Model	**Time Complexity**
Uncertainty Simple Perceptron	$O(n)$
TranAD (Transformer)	$O(n^2)$
LSTM-Autoencoder	$O(n)$
GRU	$O(T \cdot (n^2 + n \cdot m))$
Isolation Forest	$O(i \cdot \psi \log \psi)$

Although the Uncertainty Simple Perceptron shows better metrics in the benchmark, it is important to understand that this model, thanks to its ability to estimate uncertainty over the data, is able to filter out a high rate of noise (approximately 82.43%) from the data it does not understand.

This allows the model to know when something is completely certain; however, it cannot predict everything, always and in all cases.

X. Implications

This model is not only useful for markets; many time series contain a high amount of noise.

A. Security

In the world of cybersecurity, it is very complex to analyze every log using machine learning. Thanks to the proposed architecture, this barrier is overcome. Throughout this paper, the drafting of an architecture was initiated in which the use of this model is proposed to identify OWASP web attacks, incorporating feature engineering techniques such as the moving standard mean to build an attribution vector for each user. This enables the detection of low-and-slow or zero-day attacks, leveraging the model's ability to compute its own uncertainty.

B. Quantum error correction

Quantum errors are one of the reasons why quantum computing has not yet reached an adoption across all industries. Large companies such as Google typically have only one logical qubit for every hundred to a thousand physical qubits. This is done because they "all operate at the same time," allowing their post-measurement state to be observed and statistical-like

methods to be used to predict the exact value. This model could help with error correction in this type of computation.

C. Critical operations

Some critical operations, such as space missions, or systems with multiple sensors like automobiles, can generate massive amounts of noise, making it difficult to predict when something will fail. This model can analyze multiple data streams and, in this way, predict whether a failure will occur or not.

XI. Modifying the Uncertainty Scale of the Model

Different values of ϵ were also evaluated to compare the model's performance. The most representative results (accuracy : coverage) were:

1) 1: $98.7\% :\approx 17.57\%$

2) 2: $\approx 93\% :\approx 38.56\%$

3) 3: $\approx 87.44\% :\approx 63.56\%$

XII. Conclusion

In this work, the "Uncertainty Simple Perceptron" is presented, the first linear model with a computational cost of O(n).

This work validates Cover's theorem. This model is the first to achieve a 98.7% accuracy rate, obtained by filtering noise in one of the world's noisiest markets (TESLA).

Finally, a cost savings greater than 99.999% was shown in inference, training, and fine-tuning costs compared to other alternatives.

References

[1] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation", 2014.
[2] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition", 1965.
[3] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", 1997.
[4] A. M. Legendre, "Nouvelles méthodes pour la détermination des orbites des comètes. Courcier", 1805
[5] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation forest", 2008
[6] F. Rosenblatt, "The perceptron, a perceiving and recognizing automaton", 1957.
[7] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain", 1958.
[8] S. Tuli, G. Casale, and N. R. Jennings, "TranAD: Deep transformer networks for anomaly detection in multivariate time series", 2022.

GitHub repository omitted for double-blind review.