Aardvark — Uncertainty Simple Perceptron for Noise and Outliers Prediction

Dylan Sutton Chavez
c.sutton.dylan@gmail.com
Mexico City, Mexico
Independent Research

Abstract

Current deep learning models (such as *CNNs*, *RNNs*, *LSTMs*, and *Transformers*) are often inefficient when dealing with noisy data or large numbers of outliers, and they also incur very high computational costs during inference and fine-tuning. In firewall applications, this makes them impractical, since it is not feasible to process every request with such expensive models. A similar issue arises in sequential data tasks with high noise levels—*such as the stocks markets, or cibersecurity*—where these models tend to break down or achieve low accuracy.

To address this problem, *Cover's theorem* was applied. According to this theorem, a linear model (in this case, a *simple perceptron*) can achieve linear separability when data are represented in a high-dimensional space, while maintaining **O(n)** time complexity. Additionally, the perceptron's step function was modified by introducing the **epsilon variable (ϵ)** to quantify uncertainty.

The proposed solution demonstrated the next performance. In sequential tasks for the stock markets, where the model achieved a **99.999% reduction in training and inference costs**, a **100% precision** while maintaining **17.6% coverage,** which is superior in high-risk environments where the cost of error is high, compared to traditional models that fail when forcing predictions in noise, trained with three years of data and one year of backtesting *(across multiple markets, like TSLA, APPL, MSFT,...)*.

Introduction

Current models composed of multiple algebraic transformations *(hidden layers)* result in time complexity greater than **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**, the mathematician **Cover (1965)** proposed — *"complex pattern classification problems, cast in a high-dimensional space nonlinearly..."*. This means that in high-dimensional projections, a simple model increases the likelihood of finding linear separability in a dataset.

Background

Linear models such as the simple perceptron **(Rosenblatt, 1957)** or linear regression **(Legendre, 1805)** share the property of making predictions within a binary window (0, 1).

These linear models operate under the "general linear combination formula" **(Legendre, 1805)**. 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.

$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

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

$$h(x) = 1\,(x \geq 0),\ 0\,(x < 0)$$

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.

$$h(x) = 1\,(x > \varepsilon),\ 0\,(x < -\varepsilon),\ 0.5\,(-\varepsilon \leq x \leq \varepsilon)$$

The uncertainty **(ϵ)** is a constant that its defined before each prediction *(creating a grey 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 **(−ϵ)**, 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)**.

To compute the model's uncertainty, the standard deviation (σ) was used across all the training epochs, multiplying the error rate (error) by a variable to control the model's sensitivity (Δ), represented as:

$$\epsilon = \sigma \,/\, (\text{error} * \Delta)$$

Learning Rule for the Simple Perceptron

For each learning epoch, the model uses the Perceptron learning rule **(Rosenblatt, 1958)** 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 **(ŷ)**, based on the binary step function's output *(0 or 1)*.

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

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 three-dimensional space that reflects momentum, the strength of price movements, and a time-series exponential component.

$$\%k = \left(\frac{close - low_n}{high_n - low_n}\right) * 100$$

$$RS = \frac{Avg.\,Gain}{Avg.\,Loss}$$
$$RSI = 100 - \frac{100}{1 + RS}$$

$$EMA_{initial} = SMA = \frac{P_1 + P_2 + \ldots + P_n}{n}$$
$$EMA_t = EMA_{t-1} + \alpha(P_t - EMA_{t-1})$$
$$\alpha = \frac{2}{n+1}$$

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

$$v = \frac{2\pi * r}{T}$$
$$sin(v),\ cos(v)$$

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

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

Training and Hardware

The model was trained and tested against TESLA's time series, using a data window starting from **November 2022, up to November 2025**. The data was divided from 2022, with **73%** used for training and the remainder for subsequent testing.

I trained the model using an Intel **i5-12450** and **32 GB of RAM**. The training was stopped at **43 epochs** due to early stopping, and the model was restored from **epoch 34** with an error rate of **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.

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 **100%**, filtering out **82.43%** of the noise.

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 (Tuli et al., 2022), LSTM Autoencoder (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014), Isolation Forest (Liu et al., 2008), and the original Perceptron.

Error Rate

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

Finally percent of incorrect predictions in the training phase.

Accuracy

Model	Accuracy
Uncertainty Simple Perceptron	100%
TranAD (Transformer)	~85-92%
LSTM-Autoencoder	~78-84%
GRU	~75-82%
Isolation Forest	~65-72%

Percentage of positive predictions in the predicted set of the model.

Training Time (approximate)

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

Time of training for the model in his training set (an estimated for each model).

Time Complexity

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

Time complexity of each model.

Prediction Abstinence (Noise for Model)

Model	Prediction Abstinence
Uncertainty Simple Perceptron	82.4274451%
TranAD (Transformer)	0%
LSTM-Autoencoder	0%
GRU	0%
Isolation Forest	0%

The abstinence metric of prediction, where the model "knows when doesn't know something".

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 87%) 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.

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 **100%** accuracy rate, obtained by filtering noise in one of the world's noisiest datasets *(TESLA)*.

Finally, a cost savings greater than **99.99999%** was shown in inference, training, and fine-tuning costs compared to other alternatives.