# Supervised Autoencoder-MLP Approach for Volatility Prediction

Ayush Singh

March 23, 2025

**Abstract**

This document provides a detailed explanation of a supervised autoencoder combined with a multi-layer perceptron (MLP) for predicting market volatility. The model uses joint training to prevent label leakage, focuses on learning low-dimensional representations that capture nonlinear behavior, and addresses data noise via regularization techniques. It includes a cross-validation scheme suitable for time-series data and a multi-label classification setup.

## 1 Introduction

Financial markets are known for their noisy and nonlinear nature, which makes forecasting volatility a challenging task. The goal is to extract features that are robust to noise yet still predictive of large movements in the market. We combine a supervised autoencoder with an MLP in a single end-to-end framework:

1. **Supervised Autoencoder:** Learns a latent representation of the input data while also considering the downstream predictive task.

2. **MLP Classifier:** Takes both the original features and the learned latent features to predict the likelihood of high-volatility events.

Crucially, the autoencoder and MLP are trained jointly in each cross-validation (CV) split so that no information from the validation set leaks into the feature extraction process.

## 2 Data Preprocessing and Cross-Validation

### 2.1 Data Cleaning and Transformations

- **Initial Cut:** Remove the first $few$ days of data. This is often done to eliminate early-phase data where variance is significantly different or incomplete.

- **Missing Data:** Forward-fill any missing values to keep the time-series structure intact without altering the sequence of events.

- **Purging and Gap:** Use a $x$-fold, $y$-gap *purged group time-series split.* In a typical time-series split, data from the future can unintentionally leak into the model if the windows are not carefully defined. A $y$-day gap helps ensure no overlap between the end of a training window and the start of the corresponding validation window.

- **Multi-Label Targets:** Transform the volatility targets into a multi-label classification problem. For example, we might have a binary label for whether the volatility crosses certain thresholds over different horizons.

- **Sample Weighting:** Weight each sample by the mean absolute response value:

$$w_i = |r_i|,$$

  where $r_i$ is the observed response related to volatility. This ensures that periods of large market moves have a bigger impact on the training process.

## 2.2 Rationale for the Cross-Validation Split

- **Avoiding Look-Ahead Bias:** Financial data has temporal structure, so a random split can cause future information to leak. The 31-day gap and purging remove any suspicious overlap.
- **Balanced Evaluation:** Using five folds allows the model to be trained and validated on multiple distinct time segments, giving a more reliable estimate of out-of-sample performance.
- **Preventing Label Leakage:** By training the supervised autoencoder only on the training fold, we ensure that feature extraction does not inadvertently use validation data.

## 3 Model Architecture

### 3.1 Supervised Autoencoder

An autoencoder consists of two main parts: the *encoder* and the *decoder*. For an input vector $x \in \mathbb{R}^n$, the encoder produces a lower-dimensional *latent representation* $z \in \mathbb{R}^m$, with $m < n$ typically.

**Encoder:**

$$z = \sigma\big(W_e\, x + b_e\big),$$

where $W_e$ is an $m \times n$ weight matrix, $b_e$ is a bias vector in $\mathbb{R}^m$, and $\sigma(\cdot)$ is a nonlinear activation function (often chosen as the swish function, $\text{swish}(x) = x \cdot \sigma(x)$).

**Decoder:**

$$\hat{x} = \sigma\big(W_d\, z + b_d\big),$$

where $W_d$ is an $n \times m$ weight matrix, and $b_d$ is a bias vector in $\mathbb{R}^n$. The reconstruction $\hat{x}$ is compared to the original input $x$ to guide how well the latent space is capturing the underlying structure.

#### Supervised Component

While a standard autoencoder only tries to reconstruct $x$, a *supervised autoencoder* also integrates the knowledge of the target labels (volatility events). This typically means the latent space is influenced by both reconstruction error and the classification loss. By doing so, we steer the autoencoder to learn features that are highly relevant to predicting volatility, rather than just compressing data blindly.

### 3.2 MLP Classifier

Alongside the autoencoder, we have a multi-layer perceptron (MLP) that performs the actual volatility classification. The MLP processes a concatenation of $[x, z]$, where $x$ is the original input and $z$ is the latent representation from the encoder.

$$h = \mathrm{MLP}\big([x, z]; \psi\big),$$

where $\psi$ includes all the parameters (weights and biases) of the MLP. Each layer in the MLP can be written as:

$$\ell_{k+1} = \sigma\Big(W_k\, \ell_k + b_k\Big),$$

and $\ell_0$ is the initial input $[x, z]$.

**Regularization Techniques**

- **Dropout:** Randomly sets a fraction of hidden units to zero during training to prevent overfitting.
- **Batch Normalization:** Normalizes intermediate layer outputs to maintain stable distributions across mini-batches.
- **Noise Injection:** Gaussian noise is often added to inputs before the encoder. This data augmentation helps the network generalize better.

## 4 Loss Function and Joint Training

We train the supervised autoencoder and MLP *together* within each fold to avoid any leakage from validation to training. The overall loss is a weighted sum:

$$\mathcal{L} = \alpha\, \mathcal{L}_{\text{recon}} + \beta\, \mathcal{L}_{\text{cls}},$$

where:

- $\alpha$ and $\beta$ control the relative importance of reconstruction versus classification.
- $\mathcal{L}_{\text{recon}}$ measures how well the autoencoder is reconstructing its input:

$$\mathcal{L}_{\text{recon}} = \frac{1}{N}\sum_{i=1}^{N}\big\|x_i - \hat{x}_i\big\|^2.$$

- $\mathcal{L}_{\text{cls}}$ is the classification loss, generally expressed via binary cross-entropy (BCE). For *multi-label* classification, this expands to summing across all labels:

$$\mathcal{L}_{\text{cls}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} w_i\Big[y_{ij}\,\log\big(h_{ij}\big) \;+\; (1-y_{ij})\,\log\big(1-h_{ij}\big)\Big],$$

  where:

  - $C$ is the number of labels (volatility thresholds or buckets).
  - $w_i$ is the sample weight for the $i$-th observation (emphasizing large market moves).

- $y_{ij}$ is the binary label (0 or 1) for the $i$-th sample and $j$-th label.
- $h_{ij}$ is the predicted probability for the $i$-th sample and $j$-th label.

Joint training means we backpropagate through $\mathcal{L}_{\text{recon}}$ and $\mathcal{L}_{\text{cls}}$ *simultaneously.* This forces the encoder to find a latent space that is not only good at reconstructing inputs but also discriminative for volatility prediction.

## 5 Additional Techniques and Implementation Details

- **Swish Activation:** $\text{swish}(x) = x \cdot \sigma(x)$. Unlike ReLU, it does not suffer from "dead neurons" and can maintain a smooth gradient flow.
- **Multiple Random Seeds:** Train the model using three different initial seeds. Averaging predictions often stabilizes the final output, reducing variance caused by random weight initialization.
- **Hyperparameter Optimization:** Use a tool like Hyperopt to systematically search for optimal layer sizes, learning rates, dropout rates, etc. This can drastically improve performance while reducing manual trial-and-error.

## 6 Why This Approach Helps With Volatility Prediction

- **Nonlinear Feature Extraction:** The autoencoder's encoder maps data into a latent space that can capture complex dynamics beyond what linear models can handle.
- **Avoiding Leakage:** By fitting both the feature extraction (autoencoder) and the classifier inside each training fold, we make sure no future or validation-set information leaks into the extracted features.
- **Robustness to Noise:** Financial time-series are messy. Injecting Gaussian noise, applying dropout, and using swish activations provide multiple layers of regularization.
- **Target-Focused Embeddings:** Because the autoencoder is *supervised*, it learns compressed features that are directly relevant for classifying high-volatility events.
- **Adaptive Weighting:** By weighting samples according to market movements, the model pays more attention to critical periods of large volatility changes.

## 7 Conclusion

This supervised autoencoder-MLP framework integrates dimensionality reduction and classification into a single training pipeline, helping to tackle the noise and nonlinearity common in financial markets. By carefully structuring the cross-validation process and jointly training the autoencoder and the classifier, we minimize data leakage and ensure that the learned representations remain focused on predicting volatility. Regularization techniques (dropout, batch norm, noise injection) further reduce overfitting, while hyperparameter tuning helps the model adapt to the nuances of real-world market data.

## 8 Potential Improvements

- **Consider Sequential Modeling:** Although the supervised autoencoder can capture complex relationships, we might also explore recurrent architectures (*e.g.*, LSTM or GRU) or transformers for potentially richer temporal feature extraction.
- **Use Residual Connections:** Adding skip connections in the autoencoder or MLP could help stabilize training and improve gradient flow, especially for deeper networks.
- **Mix Feature Sets:** Combine raw features with handcrafted signals (like technical indicators or macroeconomic variables) to enrich the input space.
- **Explore Attention Mechanisms:** An attention layer on top of the latent representation might help the model focus on the most critical time periods or features.
- **Early Stopping and Ensemble Methods:** Implement an early stopping strategy and consider ensembling multiple model checkpoints (not just multiple seeds) for more robust performance.