Implementation is here: https://github.com/shu4dev/ICS637_Final

Introduction

In the world of financial markets, it is extremely difficult to forecast how a security's price will change in the next day, next year, or even the next second. This is because numerous variables affect price fluctuations, and even a small change in one variable can dramatically impact the price of another security. Additionally, a security's price sometimes fails to reflect all available information.

In this Kaggle competition, Jane Street asked participants to build a regression model to predict values from their production systems. Their production system consists of 6 variables: data_id, time_id, symbol_id, weight, 79 anonymized features, and 9 anonymized responses. The goal of the regression model is to predict the value of responder number 6 using the 79 anonymized features. If the model can accurately predict the response value, it will help them make better decisions and discover new investment opportunities.

Dataset, Preprocessing, Split

In particular, competitors are provided with about 47 million rows of labeled data and 39 rows of unlabeled data in the form of parquet files. Again, each row consists of data_id, time_id, symbol_id, weight, 79 anonymized features, and 9 anonymized responses. The visualization of data is in Figure 1.

	date_id	time_id	symbol_id	weight	feature_00	feature_01	feature_02	feature_03	feature_04	feature_05	feature_06	feature_07	feature_08	feature_09	feature_10	feature_11
0	0	0	1	3.889038	NaN	NaN	NaN	NaN	NaN	0.851033	0.242971	0.263400	-0.891687	11	7	76
1	0	0	7	1.370613	NaN	NaN	NaN	NaN	NaN	0.676961	0.151984	0.192465	-0.521729	11	7	76
2	0	0	9	2.285698	NaN	NaN	NaN	NaN	NaN	1.056285	0.187227	0.249901	-0.773050	11	7	76
3	0	0	10	0.690606	NaN	NaN	NaN	NaN	NaN	1.139366	0.273328	0.306549	-1.262223	42	5	150
4	0	0	14	0.440570	NaN	NaN	NaN	NaN	NaN	0.955200	0.262404	0.344457	-0.613813	44	3	16

responder_0	responder_1	responder_2	responder_3	responder_4	responder_5	responder_6	responder_7	responder_8
0.738489	-0.069556	1.380875	2.005353	0.186018	1.218368	0.775981	0.346999	0.095504
2.965889	1.190077	-0.523998	3.849921	2.626981	5.000000	0.703665	0.216683	0.778639
-0.864488	-0.280303	-0.326697	0.375781	1.271291	0.099793	2.109352	0.670881	0.772828
0.408499	0.223992	2.294888	1.097444	1.225872	1.225376	1.114137	0.775199	-1.379516
-0.373387	-0.502764	-0.348021	-3.928148	-1.591366	-5.000000	-3.572820	-1.089123	-5.000000

Figure 1

Since all the variables are anonymized, the scope of data preprocessing is limited. However, there is a unique attribute to consider when evaluating model performance: all responder values from the previous date are available at the first time step of the succeeding date. In simple terms, for all instances where time_id = 0, responder values from the previous date with time_id = 0 can be used as inputs. Therefore, after day 1, the model technically has 79 features, along with the 9 responder values from the previous day as additional inputs. All the labeled data are preprocessed so that each sample also includes the responder values from the previous day. Two other preprocessing steps are performed: filling in all NaN values and adding Gaussian noise. These steps ensure the dataset is complete, with no empty cells or missing values, and the Gaussian noise helps regularize the model. Finally this dataset splits 80% for training, 10% for model evaluation and 10% for the final test.

Model, Training, Hyperparameter

It is shown (reference 1, 2) that autoencoders tend to have good performance in time-series if financial market data. Therefore, this project aims to achieve a good performance with supervised autoencoders.

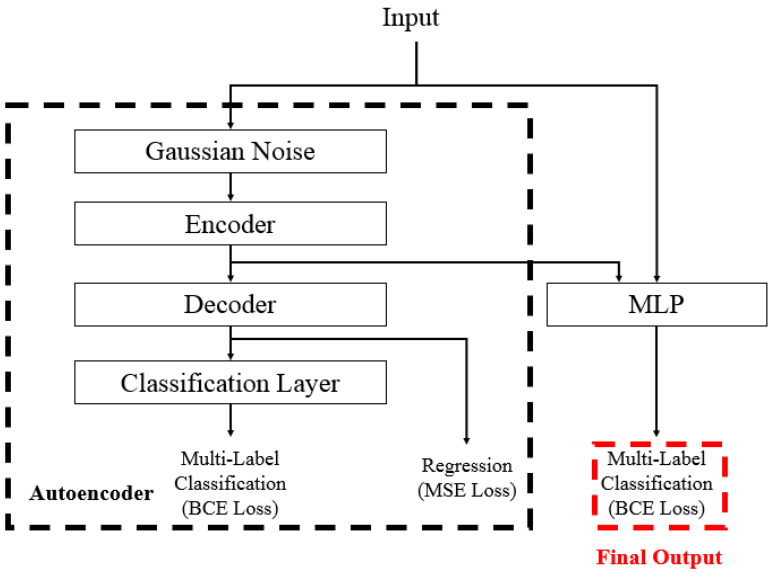

Figure 2

(Reference: https://www.kaggle.com/code/gogo827jz/jane-street-supervised-autoencoder-mlp)

Figure 2 summarizes the architecture used in this project. Instead of a classification layer and Binary Cross-Entropy loss, the model in this project is modified to use a regression layer and Mean Squared Error as the loss function for all outputs. This approach is inspired by a similar financial forecasting competition in 2021.

The core idea of this model is to train the data using two separate autoencoder branches and an additional MLP branch. The autoencoder branches produce two outputs: one from the decoder and another representing the final regression error. Finally, the model calculates the average loss from the three outputs and backpropagates it through the additional MLP branch.

```
AE(
  (encoder): Sequential(
    (0): GaussianNoise()
    (1): Linear(in_features=88, out_features=48, bias=True)
    (2): BatchNorm1d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): SiLU()
  )
  (decoder): Sequential(
    (0): Dropout(p=0.038424974585075086, inplace=False)
    (1): Linear(in_features=48, out_features=88, bias=True)
  )
  (ae_out): Sequential(
    (0): Linear(in_features=88, out_features=48, bias=True)
    (1): BatchNorm1d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): SiLU()
    (3): Dropout(p=0.42409238408801436, inplace=False)
    (4): Linear(in_features=48, out_features=1, bias=True)
```

Figure 3

```
  (main_out): Sequential(
    (0): BatchNorm1d(136, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): Dropout(p=0.10431484318345882, inplace=False)
    (2): Linear(in_features=136, out_features=448, bias=True)
    (3): BatchNorm1d(448, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): SiLU()
    (5): Dropout(p=0.49230389137187497, inplace=False)
    (6): Linear(in_features=448, out_features=224, bias=True)
    (7): BatchNorm1d(224, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): SiLU()
    (9): Dropout(p=0.32024444956111164, inplace=False)
    (10): Linear(in_features=224, out_features=224, bias=True)
    (11): BatchNorm1d(224, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): SiLU()
    (13): Dropout(p=0.2716856145683449, inplace=False)
    (14): Linear(in_features=224, out_features=128, bias=True)
    (15): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): SiLU()
    (17): Dropout(p=0.4379233941604448, inplace=False)
    (18): Linear(in_features=128, out_features=1, bias=True)
```

Figure 4

Figure 3 shows the detailed architecture of the autoencoder branch, and Figure 4 shows the detailed architecture of the MLP branch. In general, the hidden layers in the autoencoder branch consist of 48 units with a dropout rate of 0.3. The MLP branch consists of 4 layers with 448, 224, 224, and 128 units, and dropout rates of 0.1, 0.4, 0.3, and 0.2, respectively. The output of each layer is batch normalized, and the activation function used is SeLU instead of ReLU to ensure smoother gradients.

Since this is a large model with multiple hyperparameters, the results are based on the optimization library. The search space for the hidden units in the autoencoder is 48, 64, and 96. The search space for the hidden units in the MLP is 224, 448, and 512. Finally, the dropout rate is chosen from a range of 0.1 to 0.5.

The training process uses mini-batch gradient descent with a batch size of 8192 and 50 scheduled epochs, utilizing the Adam optimizer. To prevent overfitting, early stopping with a patience of 5, monitored by the validation score per epoch, is employed. As a result, the model stops training around epochs 13 to 14 to avoid overfitting.

Evaluation, Result

The evaluation metric employed is the weighted R-squared score, with the detailed formula shown in Figure 5. The weight factor is derived from the weight value included in each sample.

$$R^2 = 1 - \frac{\sum w_i(y_i - \hat{y}_i)^2}{\sum w_i y_i^2},$$

Figure 5
(Reference:https://www.kaggle.com/competitions/jane-street-real-time-market-data-forecasting/overview)

For the supervised autoencoder encoder model in this project, it achieved an R-square score of 0.1154 on the validation dataset. However, on the unlabeled test dataset, the model achieved an R-square score of 0.61 on the leaderboard (Figure 6).

Figure 6

Compared with a simple neural network and XGBoost, a simple neural network can achieve an R-square score of 0.70 on the test set, while XGBoost achieves an R-square score of 0.40 on the test set (statistics from the leaderboard).

Conclusion

In summary, this project aims to produce a model with better performance than both non-deep learning models and deep learning models with simple structures. However, the project only partially achieves this goal. It was expected that the model would perform similarly on the test set as it did on the validation set. One possible explanation is that the financial market data is still too dynamic for the model to generalize the patterns of the features effectively. Another observed difference is that historical research on autoencoders has primarily focused on classification tasks rather than regression tasks. To address these issues, future improvements could include training the model with a more regularized technique to determine if it can better approximate the patterns or redesigning the problem by labeling each sample, thereby transforming it into a classification task.

Reference

1. Bieganowski, B., & Ślepaczuk, R. (2014). *Supervised Autoencoder MLP for Financial Time Series Forecasting*. Arxiv.org. https://arxiv.org/html/2404.01866v1
2. Hosseini, S. A., Hosseini, S. A., & Houshmand, M. (2024). Variational autoencoder-based dimension reduction of Ichimoku features for improved financial market analysis. *Franklin Open*, *8*, 100135. https://doi.org/10.1016/j.fraope.2024.100135
3. Reference of model architecture: https://www.kaggle.com/code/gogo827jz/jane-street-supervised-autoencoder-mlp