Neural ODEs and Wasserstein gradient flows in embedding space

Sanjit Dandapanthula
Department of Statistics
Carnegie Mellon University
Pittsburgh, PA 15213
sanjitd@cmu.edu

Abstract

In this paper, we study a novel approach for stock price prediction using a flow-based model, inspired by previous work treating transformers as a discrete-time approximation to a continuous time flow. Our model is based on learning a flow pushing forward a discrete measure over signal embeddings forward to the next day. This model has strong empirical foundations due to the recent success of the transformer architecture. Furthermore, our model has a strong theoretical foundation, since the flow model that we choose can be interpreted as an interacting particle system or a Wasserstein gradient flow. We show that the flow model is able to achieve a statistically significant positive correlation with the true return, and that it is much more interpretable than a black-box neural network model.

Contents

1 Introduction

The problem of stock return prediction is extremely important for financial firms, and our approach gives a novel methodology for any prediction problem, which is interesting in its own right. In this paper, we study a novel approach for stock price prediction using a flow-based model, inspired by the work of Geshkovski et al. [4] studying transformers as a discrete-time approximation to a

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

continuous time flow; we begin with some background on this work. Geshkovski et al. [4] show that the transformer architecture with layer normalization can be interpreted as a continuous time flow map on $(S^{d-1})^n$ (the unit sphere in $\mathbb{R}^d$), where d is the embedding dimension and n is the context size. Given a sequence of n tokens embedded in $\mathbb{R}^d$, layer normalization forces the embeddings $\{x_i\}_{i=1}^n$ to lie on the unit sphere S^{d-1}. Then, one can construct the measure $\frac{1}{n}\sum_{i=1}^n \delta_{x_i}$ on S^{d-1}, where x_i is the ith embedded token. Let $Q(t)$, $K(t)$, and $V(t)$ denote the query, key, and value matrices at time $t \geq 0$. Geshkovski et al. [4] focus on the attention mechanism, and show that the action of a transformer on an input sequence $(x_i(0))_{i=1}^n$ can be modeled using the dynamics

$$x_i'(t) = \mathrm{proj}_{\mathrm{span}(x_i(t))^\perp}\left(\frac{1}{Z_{\beta,i}(t)}\sum_{j=1}^n e^{-\beta\langle Q(t)x_i(t),\, K(t)x_j(t)\rangle}\, V(t)\, x_j(t)\right), \tag{1}$$

where

$$Z_{\beta,i}(t) = \sum_{j=1}^n e^{-\beta\langle Q(t)x_i(t),\, K(t)x_j(t)\rangle}$$

is the partition function and $\beta \in \mathbb{R}$ is the interaction strength. In general, $Z(t)$ is a nontrivial object of study, but Geshkovski et al. [4] note that with the approximation $Z(t) \approx n$ and $Q(t) = K(t) = V(t) = I_d$, the dynamics behaves similarly but simplifies to the surrogate model

$$x_i'(t) = \mathrm{proj}_{\mathrm{span}(x_i(t))^\perp}\left(\frac{1}{n}\sum_{j=1}^n e^{\beta\langle x_i(t),\, x_j(t)\rangle}\, x_j(t)\right).$$

Now, this dynamics is that of an interacting particle system, with pairwise interactions between each of the embedding vectors. Let $\mathcal{P}(S)$ denote the set of probability measures over S. The surrogate model is then a Wasserstein gradient flow with energy functional $E : \mathcal{P}(S^{d-1}) \to \mathbb{R}$ given by

$$E_\beta(\mu) = \frac{1}{2\beta}\int e^{\beta\langle x,\, y\rangle}\, d\mu(x)\, d\mu(y).$$

This energy functional will then be decreasing along the flow of the surrogate model. Geshkovski et al. [4] then show that the surrogate model exhibits (and therefore explains) the clustering phenomenon often found in transformers. Although Geshkovski et al. [4] do not study the dynamics of the linear layer typically used for the final prediction in a transformer model, they interpret the transformer as a continuous time flow in the space of embeddings. Inspired by this paper, we are interested in stock price prediction by training a flow model for feature embeddings in continuous time.

Before discussing the details of the model, we first list several advantages that such a model might have over traditional neural network prediction models.

1. **Interpretability:** The flow-based model is extremely interpretable in the sense that we learn the continuous time dynamics moving signal embeddings through time. Analyzing this flow allows us to understand the underlying mechanisms driving the predictions, which is often a challenge with traditional neural networks. For instance, one can analyze the direction in which each signal is moving (in embedding space), in continuous time.
2. **Flexibility:** The flow-based model can be used to determine a continuous time trajectory of the signals, which can allow us to make predictions at any intermediate time. In this sense, a flow-based model is the infinite-dimensional analog of a traditional neural network.
3. **Robustness:** The flow-based model is robust to local irregularities in the data and sensitive to the global geometry of the loss landscape; for instance, Chizat and Bach [3] show that under some regularity conditions, the Wasserstein gradient flow converges in the limit to the global minimizer of the energy functional, even in the presence of local minima.
4. **Uncertainty quantification:** If we have regions of uncertainty for each of the signals in the dataset, we can use the flow model to easily propagate this uncertainty through the model at any intermediate time.

Furthermore, from a mathematical perspective, it is novel and interesting that such a model can even be trained for a simple supervised learning task. Now, we are ready to discuss the details of our

problem and the model that we will use. In the Trexquant numerical dataset, we are trying to predict the returns of several stocks given 200 relevant alpha signals. We have samples $\{(x_i, y_i)\}_{i=1}^{2247484}$, where $x_i \in \mathbb{R}^{200}$ are signals and $y_i \in \mathbb{R}$ are scalar returns. Each of these samples is associated with a stock index si_i and a day index di_i.

We begin by splitting the dataset into the earliest 80% for training and the later 20% for testing (sorted by day index); this way, there is no leakage between the training and testing sets. In order for the features to be on a similar scale, we first fit a standard scaling to the training set of x_i, subtracting the empirical mean and dividing by the empirical standard deviation of each feature.

1.1 Data preprocessing

We split the data chronologically, using the first 80% of the data for training and the last 20% for testing. We do this in order to ensure that there is no train-test leakage; if the testing data were interspersed temporally with the training data, our model could potentially interpolate between training points to achieve good performance on the test set. We then standardize the features x_i by subtracting the empirical mean and dividing by the empirical standard deviation of each feature (fitted on the training set). This is done to ensure that all features are on a similar scale, which is important to ensure stability of the training process. For instance, we fit a shared embedding model across all signals, so if one signal is on a different scale than the others, it could dominate the training process. We observe that the signals are mostly normally distributed (with some sparsity), so this standardization is appropriate for our dataset.

1.2 Computing embeddings

First, we would like to embed each individual feature into $\mathbb{R}^{32}$, in a way which preserves some of the feature's structure. In order to compute embeddings, we train an autoencoder on the training set to predict the next day's features (for the same stock index) given the current day's features. The encoder is trained in the following way: we use a shared multi-layer perceptron $f_\theta : \mathbb{R} \to \mathbb{R}^{32}$ to embed a feature into $\mathbb{R}^{32}$. The architecture that we use is a linear layer mapping into $\mathbb{R}^{64}$, followed by ReLU activation and another linear layer mapping into $\mathbb{R}^{32}$. Furthermore, we maintain a learnable positional embedding $p_\theta^{(j)} \in \mathbb{R}^{32}$ for each of the 200 features and a learned stock embedding $s_\theta(\mathrm{si}_i)$, which are added to the output of the encoder. For a particular stock on a particular day, we compute the embedding of the jth signal as

$$\hat{e}_i^{(j)} = \mathrm{proj}_{S^{31}}\left(f_\theta(x_i^{(j)}) + p_\theta^{(j)} + s_\theta(\mathrm{si}_i)\right),$$

normalizing to project our embedding onto the unit sphere. After embedding, we train a multi-layer perceptron $g_\theta : \mathbb{R}^{200\times 32+1} \to \mathbb{R}^{200}$ to predict the next day's signals given the current day's signal embeddings and the number of days (the gap) until the next observation of the same stock. The architecture of this model is a linear layer mapping into $\mathbb{R}^{512}$, followed by ReLU activation and another linear layer mapping into $\mathbb{R}^{200}$. The model is trained using the mean squared error loss between the predicted and actual next day signals. We train our model for 30 epochs using the Adam optimizer with a learning rate of 3×10^{-4} and a batch size of 512. Using these embeddings, we compute the empirical measure

$$\mu_i = \frac{1}{200}\sum_{j=1}^{200} \delta_{\hat{e}_i^{(j)}},$$

which we flow forward using the dynamics described in Equation 1.

1.3 Training the flow model

The flow model is learned using the ODE dynamics of the surrogate model given in Equation 1, and is an example of a neural ODE as described by Chen et al. [2]. We assume that $Q(t)$, $K(t)$, and $V(t)$ are all constant matrices across time. Therefore, it remains to learn Q, K, V, and the interaction strength parameter $\beta > 0$. We will later observe empirically that flowing the features does still give significant performance improvements over a standard multi-layer perceptron predicting y from the embeddings induced by x. Starting with the embeddings $\hat{e}_i$ of the input features, recall that we

constructed the empirical measure

$$\mu_i = \frac{1}{200}\sum_{j=1}^{200} \delta_{\hat{e}_i^{(j)}}$$

on $\mathbb{R}^{32}$. Let $T_0 = 0.05$ denote a small constant, this is a hyperparameter of our algorithm, representing the amount of time to flow forward, per day. We then flow the measure forward under the dynamics given in Equation 1 for $T_0 \times n_{\text{days}}$ time, using a Runge-Kutta method of order 4 (RK-4), for four time steps. We backpropagate through the ODE solver to learn the parameters Q, K, V, and β using the adjoint method, as described in Chen et al. [2]. In order to train the flow model, we use the cosine loss between the flowed-forward feature embeddings and the actual observed next-day embeddings. Denote the jth flowed-forward embedding at row i by $T_\theta(\hat{e}_i^{(j)})$. We use the cosine loss, defined as

$$\ell(\mu) = \sum_{j=1}^{200}(1 - \langle T_\theta(\hat{e}_i^{(j)}), \hat{e}_{i+1}^{(j)}\rangle).$$

Note that the flow and all embeddings are restricted to the unit sphere S^{32-1}, so the cosine loss encourages the flowed-forward embeddings to be close to the next-day embeddings in the natural metric on the sphere; this is a natural choice of loss function, since the cosine distance is invariant to the scale of the embeddings.

In a previous attempt, we used the *Chamfer loss*, defined as

$$\ell_{\text{Chamfer}}(\mu,\nu) = \frac{1}{2}\left(\int_{\text{supp}(\mu)} \inf_{y\in\text{supp}(\nu)} \|x-y\|_2^2 \, d\mu(x) + \int_{\text{supp}(\nu)} \inf_{y\in\text{supp}(\mu)} \|x-y\|_2^2 \, d\nu(x)\right).$$

The Chamfer loss between two discrete measures is clearly an upper bound on the squared Wasserstein 2-distance $W_2(\mu,\nu)^2$, so a bound on the Chamfer loss implies a bound on the Wasserstein 2-distance between two measures. Hence, this choice of loss function is robust in the same way that the Wasserstein 2-distance is. For more details on the Chamfer distance, see Butt and Maragos [1]. However, we found better results using the cosine loss.

1.4 Prediction of returns

Finally, we fit a simple multi-layer perceptron predicting the return y_i from the (flattened) flowed-forward embeddings $T(\hat{e}_i^{(j)}) \in \mathbb{R}^{200\times 32}$ obtained from the flow model. The architecture of this model is:

- Linear layer mapping into $\mathbb{R}^{128}$
- Batch normalization, ReLU activation, and dropout ($p = 0.3$)
- Linear layer mapping into $\mathbb{R}^{64}$
- Batch normalization, ReLU activation, and dropout ($p = 0.3$)
- Linear layer mapping into $\mathbb{R}$.

We train this model using the mean squared error loss between the predicted and actual returns. We use the Adam optimizer with a learning rate of 10^{-3} and a batch size of 256. The model is trained for 10 epochs.

2 Results

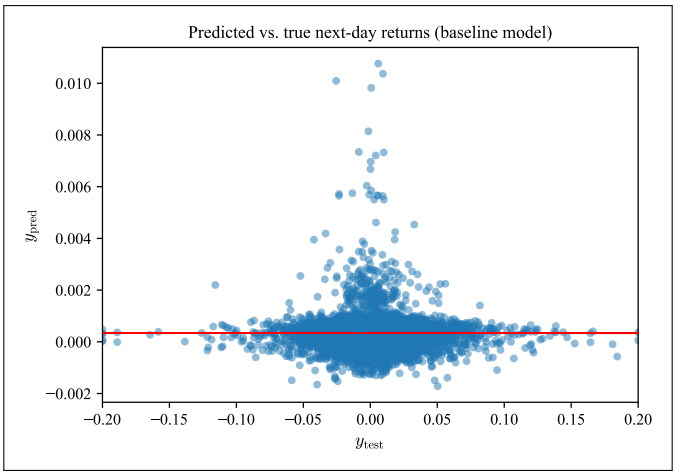
All of our code can be found at

`https://www.github.com/sanjitdp/gradient-flow-stocks`.

In this section, we verify empirically that adding the flow model to the pipeline does improve our predictions of the next-day return. Recall that we used (chronologically) the first 80% of the data to train our model and the last 20% to test it. If we fit an MLP predicting y_i directly on the flattened embeddings $\hat{e}_i$ (without the flow model), we obtain the baseline results shown below on the test set:

Pearson-r correlation coefficient	-0.0006
p-value	0.9491

These are the resulting predictions on the test set:

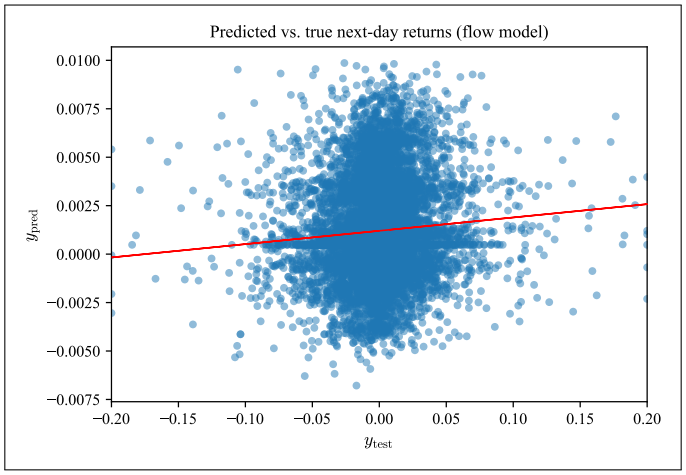


Naive MLP predictions of the next-day return.

This is a very weak correlation, so we would hope that the flow model accounts for more of the variance in the data. We observe from this example that the MLP is unable to learn any useful patterns from the signals, which mostly contain noise. In particular, the naive MLP learns to predict a value near zero most of the time, without explaining much of the variance in the dataset. Adding the flow model followed by an MLP predicting y_i from the flowed-forward embeddings gives the following results on the test set:

Pearson-r correlation coefficient	0.0901
p-value	< 0.0001

Hence, our flow-based model is able to learn useful patterns which generalize much better to the training dataset. These are the resulting predictions on the test set:



Flow model based MLP predictions of the next-day return.

Hence, we have demonstrated that even a very simple flow model allows us to make nontrivial and statistically significant predictions of the next day return, as evidenced by the small p-value and nontrivial correlation. Furthermore, as mentioned in the introduction, the flow model is much

more interpretable than a black-box neural network model, and allows us to analyze the underlying dynamics of the features and evaluate the model at any intermediate time. While the directional accuracy of the model is not very high, we note that the outputs of this model can be post-processed and used to inform a trading strategy.

3 Future work

Although our results show that our model is already able to achieve a significant positive correlation with the true returh, in this section, we discuss potential future work that could be done to improve the model. One potential avenue for improvement is to simultaneously train the embedding, flow, and prediction models. In our current implementation, we freeze the weights after each step and train the next model. However, it is possible that the flow model would benefit from being trained jointly with the embedding and prediction models; this would require a more careful choice of neural network architecture to avoid vanishing gradients.

Another potential avenue for improvement is to use more steps in the RK-4 ODE solver. In our current implementation, we only use 4 steps due to computational constraints, but it is possible that using more steps would allow us to better approximate the dynamics of the underlying flow.

Yet another potential avenue for improvement is to maintain a history of the previous flow for a stock. In our current implementation, we only use the current day's flow to predict the next day's return. However, it may improve the performance of the model for the current day's flow to use some momentum from the previous day's flow (since the dynamics of the signals are theoretically continuous).

Finally, it is possible that allowing Q, K, and V to vary with time would allow the model more flexibility in representing the flow. For instance, one could assume that Q, K, and V are periodic functions of time, which is analogous to using an infinitely deep transformer with periodic attention heads. This would allow the model to learn a more complex flow, but would also increase the number of parameters to be learned.

4 Conclusion

In this paper, we have presented a novel approach for stock price prediction using a flow-based model in the space of embeddings. We have shown that the flow model is able to achieve a statistically significant positive correlation with the true return, and that it is much more interpretable than a black-box neural network model. Furthermore, we have discussed several potential avenues for improvement, including joint training of the embedding, flow, and prediction models, using more steps in the RK-4 ODE solver, maintaining a history of the previous flow for a stock, and allowing Q, K, and V to vary with time.

5 Acknowledgments

The author thanks Sierra Stevenson for allowing the use of her computer to train the models described in this paper.

References

[1] M Akmal Butt and Petros Maragos. Optimum design of chamfer distance transforms. *IEEE Transactions on Image Processing*, 7(10):1477–1484, 1998.

[2] Ricky Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[3] Lenaic Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. *Advances in neural information processing systems*, 31, 2018.

[4] Borjan Geshkovski, Cyril Letrouit, Yury Polyanskiy, and Philippe Rigollet. A mathematical perspective on transformers. *arXiv preprint arXiv:2312.10794*, 2023.