PAIRS TRADING WITH LONG-SHORT TERM MEMORY

Master’s Thesis
Saska Karsi
Aalto University School of Business
Finance
Spring 2019
Author  Saska Karsi
Title of thesis  Pairs trading with long-short term memory
Degree  M.Sc.
Degree programme  Finance
Thesis advisor(s)  Matti Suominen
Year of approval  2019  Number of pages  43  Language  English

Abstract

The cross-disciplinary paper explores the applicability of different neural network architectures in financial time series prediction, bridging the still rather wide gap between machine learning and finance. Three network architectures of increasing complexity - feedforward, recurrent and long-short term memory - are presented while justifying why the additional model complexity is necessary for market data. A long-short term memory model is then chosen according to the requirements of the data characteristics. The specific input variables of the model are then chosen based on existing pairs trading literature and the model is implemented and iteratively tuned to fit a previously explored pair, predicting its spread. The architecture is then validated on a larger sample picked via a cointegration metric before finally eliminating look-ahead bias by testing on a new period on a new subset of stocks. The computationally inexpensive model captures effects associated with pairs trading observable throughout the sample and forms a portfolio with large excess returns significant through early 2019, suggesting that excess returns can still be generated within comoving stocks using advanced nonlinear methods.

Keywords  Neural networks, pairs trading, machine learning
Pairs trading with long-short term memory

Saska Karsi

May 31, 2019

Table of Contents

1 Introduction 2
2 Theoretical background 3
  2.1 Feedforward neural networks 3
  2.2 Recurrent neural networks 6
  2.3 Long short-term memory 8
  2.4 Characteristic issues with market data and methods of mitigation 12
  2.5 Pairs trading 13
  2.6 Cointegrated augmented Dickey-Fuller test 14
3 Empirical Analysis 14
  3.1 Case UPM-Stora Enso 15
  3.2 CRSP 2010-2013 19
  3.3 CRSP 2018-2019 21
4 Discussion 23
5 Conclusions 25
  5.1 Research summary 25
  5.2 Practical implications 25
  5.3 Limitations of the study 26
  5.4 Further research 26
6 References 28
7 Appendices 30
  7.1 Appendix A: LSTM implementation 30
  7.2 Appendix B: Other financial problems for neural networks 35
  7.3 Appendix C: Portfolio-specific returns and volatilities for UPM-Stora 37
  7.4 Appendix D: Additional tables and figure for CRSP 2010-2019 38
  7.5 Appendix E: Case AMD-Intel 41
1 Introduction

If there exist data-driven investment decision makers, there exists a function that maps from related and utilized data to a portion of the flow of capital into a security in a market. When security prices are demand-driven, finding an approximation can yield insight, and assuming there exists a lag from data availability to investment decision it can also have predictive power. Learning this price function is naturally unfeasible, but perhaps capturing a significant enough part of it with a universal approximator to gain abnormal returns is not. Neural networks, according to Hornik et al (1989), could be this universal approximator.

As computational power, the availability of large amounts of data and our understanding of neural networks continue to grow, these tools are used to help solve increasingly complex problems. Financial literature almost seems a rare exception - published results on machine learning are still few and far between. Sequence models in particular have taken great leaps in recent years. As the name implies, these models are by design ideal for processing time series. One example of such a model is long-short term memory introduced by Hochreiter & Schmidhuber (1997). Published successful applications of long-short term memory networks (LSTMs) in financial literature are still scarce, but increasing each passing day when researchers look for more sophisticated alternatives to traditional regressions. With the narrow adoption these methods have great potential for generating alpha.

The purpose of this paper is to highlight some of the problems with neural networks in financial time series analysis with possible solutions to those problems as well as present an implementation and application of a long-short term memory model to a prediction problem. Long-short in the context of the network is intended to imply it utilizes a few tricks to make long-term memory in recurrent neural networks behave like short-term memory. These tricks are further explained in section 2.3, but I feel it is important to clarify the clash in terminology, as the term has little to do with the long-short portfolio we will use it to construct. The paper contributes to existing literature by predicting a phenomenon with a model that hasn’t yet been applied to it, in a period where simpler methods have lost their performance. Another contribution is the feasibility study, which should help explain the relevance of the model architecture in terms of market prediction. Both contributions are cross-disciplinary and help bridge the gap between machine learning and finance.

The structure of the paper is as follows. First we provide an overview on neural net-
works as a predictive tool and provide justification for our choice of architecture, the
LSTM, for market data in particular by first exploring simpler architectures and high-
lighting issues in their behavior considering some characteristics of market data. Next
we review relevant financial literature. We then implement and use the tool for em-
pirical analysis utilizing effects similar to those used in pairs trading (a market-neutral,
comovement-reliant investment strategy) before critically examining our results and fi-
nally concluding.

2 Theoretical background

This section is a cross-disciplinary literature review first explaining three network archi-
tectures of increasing complexity. The increased complexity is justified by demonstrating
the insufficiency of a simpler model. We then introduce some challenges in market data
and review the state of financial literature related to our analysis before finally introduc-
ing a statistical test we use for stock picking.

2.1 Feedforward neural networks

Artificial neural networks (ANNs) are computing systems that, as stated previously, can
under the right circumstances and design function as universal approximators (Hornik et
al, 1989). This effectively means that with enough computational resources they can ap-
proximate any function. A feedforward neural network is the simplest form of an ANN,
where recurrent connections (explained in section 2.2) are not present. A feedforward
neural network consists of layers of linear and nonlinear functions.

In its simplest form, a 1-layer neural network can be formulated as $a = f(Wx + b)$
where $f$ is called an activation function. To a reader versed in financial literature this
closely ties in with a more familiar model, as the model above without the activation
function is equivalent to a matrix-notated linear regression, provided $W$ is a row vector.
The activation function is introduced to induce nonlinearity, the importance of which is
explained in section 2.1.4 (Properties of a good activation function). Multiple of these
layers can then be stacked together so that $a_i = f(W_i a_{i-1} + b_i)$. A feedforward network
can then intuitively be thought of as a stack of nonlinear regressions where the output
of the previous regression is the input of the next. Most neural networks today follow
a forward-backward propagation structure for learning, which we will explain next. We
will then explain some terminology before discussing activation function properties.

2.1.1 Forward pass

Activating, that is extracting an output from, a neural network is done by propagating an input forward through each function in the network. This is aptly called forward propagation or the forward pass. The output of the $n$-layered network is the activation of the last layer, $a_n$. This activation can be used as a prediction as-is or through an intermediary function so that $\hat{y} = f(a_n)$. Note that there is no reason this function couldn't be another neural network, allowing multiple networks of different architectures to be stacked together.

2.1.2 Backward pass

Fitting the network to historical data requires us to define a loss (or utility) function $L(y, \hat{y})$ where $y$ is the target variable or true value and $\hat{y}$ is our forward propagation output. We then compute the gradient of each weight matrix w.r.t. that loss function $\frac{\partial L}{\partial W_i}, \{i \in \mathbb{Z} \mid i \leq n\}$, usually via a computational graph utilizing the chain rule. The weights $W_i$ must then be updated accordingly. This is called backpropagation or the backward pass, a way to implement gradient descent (or hill climb) in a network. Gradient descent minimizes the loss function by updating the weights in the direction of the negative gradient. Hill climb does the opposite to maximize a utility function. An overview of different types of gradient descent algorithms can be found in (Ruder, 2016). For the loss function in our networks we will use L2 loss $\sum_{i=1}^{n} (y_i - \hat{y_i})^2$, closely related to mean squared error.

2.1.3 Terminology

Hyperparameters. The term hyperparameter in machine learning (not to be confused with hyperparameter in Bayesian statistics) refers to any parameter that is set prior to training. This can be for example the dimensionality of the weight matrices in the network, the batch size (number of observations input simultaneously) or the learning rate at which the gradient descent update is applied. Hyperparameters can be manually tuned or optimized algorithmically. Different machine learning algorithms, models and architectures have their own (often distinct) sets of hyperparameters with varying levels of importance (effect on performance) called tunability (Probst et al., 2018).
**Epoch.** An epoch is one complete training pass through the entirety of the training data set. For a data set of 1000 examples with a batch size of 100 an epoch would be 10 passes, each containing forward and backward propagation.

**Initialization.** Initialization refers to how the first state of the network parameters is, most often randomly, formed. Parameter initialization methods can have significant effects on convergence (e.g. Sutskever et al., 2013). In our final model we use the so called Xavier initialization (Glorot and Bengio, 2010), which aims to initialize the weight matrices so that $W_{ij} \sim N(0, \frac{1}{n})$.

**Training, validation and test sets.** The training set is a set of data that is used to fit the model, the validation set is used for evaluating the out-of-sample performance of the model during training time and the test set is used for evaluating the final model fit. The validation set can also be used as a form of regularization, a method used to improve out-of-sample performance further explained in section 2.4.1, by stopping the training early. The test set and the validation set are separate to reduce look-ahead bias, which decision making according to test set performance during training can induce.

### 2.1.4 Properties of a good activation function

While technically arbitrary, activation functions should have their properties subject to careful consideration. We will now present two of the desirable properties, differentiability and nonlinearity.

Firstly, an activation function should be differentiable. This is required for backpropagation, and using another method reliant on calculating the numerical gradient for each layer is usually prohibitively expensive. Differentiability over the domain is not always technically accurate, e.g. in the case of rectified linear units which are non-differentiable at 0, but a single point of indifferentiability is seen as a negligible issue.

Secondly, the function should be nonlinear. For an $n$-layered artificial neural network using e.g. the identity function, $a_n = (W_n, ..., W_1)x + b_n + \Sigma_{i=1}^{n-1}W_{n, i+1}b_i$ can also be written as $a = W_p x + b_p$ where $W_p = (W_n, ..., W_1)$ and $b_p = b_n + \Sigma_{i=1}^{n-1}W_{n, i+1}b_i$. It becomes apparent that an $n$-layered network with no nonlinear activations is equivalent to a single layer network and cannot learn nonlinear mappings.

We will not go deeper into activation function choice, but it bears mentioning that all
activation functions in our final model are nonlinear, finite range (with a small output interval, so-called “squashing functions”) and continuously differentiable.

2.1.5 Drawbacks in market data

The feedforward neural network can only account for temporality in data by input vector concatenation, which can drastically increase the dimensionality of both the data and the weight matrices. This leads to poor performance in relation to the computational and memory cost of the model. The dimensionality of a single observation also needs to be finite and known at initialization time meaning the architecture is poorly equipped to address variable length inputs. Both of these issues are addressed by the recurrent neural network.

2.2 Recurrent neural networks

A recurrent neural network, or RNN, is very similar to a feedforward neural network, but on each iteration the previous activation, or hidden state, is passed into the network. This is called a recurrent connection and separates RNNs from feedforward networks, creating a natural extension better suited to process sequences. The recurrent connection can intuitively be thought of as the memory of the network. Whereas the feedforward network would take in a temporally dependent input of 100 observations with 8 variables as an input of size 800, the RNN allows us to process inputs of size 8 sequentially 100 times. This is computationally more efficient and the sequence length is not fixed pre-initialization.

In general, any data-target pair fed into an RNN will have to be a three-dimensional array in the shape (number of time steps, number of examples, number of variables) – note that there is no reason why a network output could not be multidimensional.

2.2.1 Forward pass

The forward pass of a recurrent neural network is given by

\[
a_t = \tanh(W^x_t x_t + W^a_{a^{t-1}} + b),
\]

(1)
where it is apparent that the iteration at the previous time step has a direct effect on all consequent steps. The weight matrix of the recurrent connection $W_a^a$ controls the flow of information through time. A familiar analogue could be to think of $W_a^a$ as an autoregressive parameter. It is, however, constant through time; this leads to a fundamental flaw called the vanishing gradient problem.

### 2.2.2 Backward pass and the vanishing gradient problem

With an RNN, the derivatives calculated when backpropagating will also have to be calculated w.r.t. previous states in the so-called *backpropagation through time*. A critical issue faced when doing this in RNNs is the so-called vanishing (exploding) gradient problem. Since the network shares its weights with all other time steps, values exponentially vanish (explode) outside the feasible range of the finite-precision floating point arithmetic used by computers. The derivative of an activation $a_t$ w.r.t the previous $a_{t-1}$ is given by

$$\frac{\partial a_t}{\partial a_{t-1}} = W_a^{aT}(1 - \tanh(W_a^x x_{t-1} + W_a^a a_{t-1} + b)^2),$$

and thus w.r.t. $a_{t-2}$ is

$$\frac{\partial a_t}{\partial a_{t-1}} \frac{\partial a_{t-1}}{\partial a_{t-2}} = W_a^{aT}(1 - \tanh(W_a^x x_{t-1} + W_a^a a_{t-1} + b)^2) W_a^{aT}(1 - \tanh(W_a^x x_{t-2} + W_a^a a_{t-2} + b)^2).$$

Continuing via the chain rule we obtain

$$\frac{\partial a_t}{\partial a_{t-n}} = \frac{\partial a_t}{\partial a_{t-1}} \ldots \frac{\partial a_{n+1}}{\partial a_{t-n}} = W_a^{aT} (1 - c_{t-1}^2) \ldots \cdot W_a^{aT} (1 - c_{t-n}^2)$$  \hspace{1cm} (2)

where

$$c_{t-i} = \tanh(W_a^x x_{t-i} + W_a^a a_{t-i} + b).$$

This raises a substantial problem for capturing long-term dependencies, which market data often contains (e.g. Rozef and Kinney, 1976). Consider equation (2) when $W_a^a$ is a square matrix with eigendecomposition $AA^{-1}$. Propagating a one-year seasonality dependency from daily data would cause the elements of the diagonal on the first time
step w.r.t. the last to be $b^{252}$. Any number below 1 on this diagonal will vanish, and above explode. The next model, long short-term memory, was designed largely to mitigate this issue.

### 2.3 Long short-term memory

Long short-term memory networks or LSTMs, first introduced by Hochreiter & Schmidhuber (1997) are a subclass of RNNs that mitigate a crucial issue with the traditional architecture: the vanishing or exploding gradient problem.

Figure 1 presents a diagram of an LSTM cell, equivalent to equations (3-8) below, where square boxes represent activation functions containing dot products and circles are functions applied element-wise. As we can see from the two parallel horizontal lines, both the cell state $c_{t-1}$ and the activation $a_{t-1}$ of the previous state is propagated into the cell at $t$ and updated before propagating them to an identical cell at $t + 1$. It should be noted however that network parameters are shared between cells, so the cell feeding back into itself is equally valid intuition. It seems somewhat conventional for authors working with these networks to create their own confusing diagram.

Equations in the following sections are largely derived for purposes of implementation and illustration. The Python implementation of the functions presented in this section can be found in appendix A.
2.3.1 Forward pass

The forward pass of an LSTM cell is given by the equations

\[ \hat{c}_t = \tanh(W_c [a_{t-1}, x_t] + b_c), \]  
(3)

\[ u_t = \sigma(W_u [a_{t-1}, x_t] + b_u), \]  
(4)

\[ f_t = \sigma(W_f [a_{t-1}, x_t] + b_f), \]  
(5)

\[ o_t = \sigma(W_o [a_{t-1}, x_t] + b_o), \]  
(6)

\[ c_t = u_t \circ \hat{c}_t + f_t \circ c_{t-1}, \]  
(7)

and

\[ a_t = o_t \circ \tanh(c_t), \]  
(8)

where \( c \) is the cell state, \( u_t, f_t \) and \( o_t \) are the update, forget and output gates respectively, \([a, b]\) denotes the concatenation of matrices \( a \) and \( b \), \( \circ \) denotes the Hadamard product, \( \tanh \) is the hyperbolic tangent function

\[ \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \]

and \( \sigma \) is the sigmoid function

\[ \sigma(x) = \frac{e^x}{e^x + 1}. \]  
(9)

Both the hyperbolic tangent and the sigmoid function are applied element-wise.

Equations (4-6) are known as the gates of the LSTM cell. They control the persistent cell state \( c_t \) in a specific way. In (3), \( \hat{c}_t \) is known as the candidate cell state. The update
gate \( u_t \) controls which elements of \( \tilde{c}_t \) are passed to \( c_t \). The forget gate \( f_t \) in (5) allows for a straight connection from the concatenation of the previous activation and new input for specific lags. These lags are not predetermined, but rather learned by the network, as explained in the next section. These relationships can also be seen in figure 1.

Equation (7) is crucial for our purposes for the formation of \( \frac{\partial c_t}{\partial c_{t-1}} \). The exact mechanics are too involved for this paper. Refer to (DiPietro et al., 2017) and (Hochreiter & Schmidhuber 1997).

2.3.2 Backward pass

Partial w.r.t. the cell state after application of the forget gate is

\[
\frac{\partial L}{\partial c_t} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_t} \circ o_t \circ (1 - \tanh(c_t)^2) + \frac{\partial L}{\partial c_{t+1}} \circ f_t.
\]

and thus w.r.t. to the previous cell state

\[
\frac{\partial L}{\partial c_{t-1}} = \left( \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_t} \circ o_t \circ (1 - \tanh(c_t)^2) + \frac{\partial L}{\partial c_{t+1}} \circ f_t \right) \circ f_t.
\] (10)

Note that this is a simplification, since \( f_t \) is also functionally dependent on \( c_{t-1} \). It is, however, a crucial part of how the LSTM avoids the vanishing gradient problem. Recall from equations (5, 9) that the output of \( f_t \) is in \((0, 1)\). Intuitively this output can then be thought of as the importance, or weight, the network assigns to a given lag.

The derivatives of the gates are

\[
\frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_t} \circ o_t \circ (1 - o_t),
\]

\[
\frac{\partial L}{\partial u_t} = \left( \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_t} \circ o_t \circ (1 - \tanh(c_{t+1})^2) + \frac{\partial L}{\partial c_{t+1}} \circ \tilde{c}_t \circ u_t \circ (1 - u_t), \right) \circ f_t \circ (1 - f_t),
\] (11)

\[
\frac{\partial L}{\partial f_t} = \left( \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_t} \circ o_t \circ (1 - \tanh(c_{t+1})^2) + \frac{\partial L}{\partial c_{t+1}} \circ c_{t-1} \circ f_t \circ (1 - f_t), \right) \circ f_t \circ (1 - f_t),
\] (12)

and
\[
\frac{\partial L}{\partial \tilde{c}_t} = \left( \frac{\partial L}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial a_t} \circ o_t \circ (1 - \tanh(c_{t-1}))^2 + \frac{\partial L}{\partial \tilde{c}_{t+1}} \circ u_t \circ \tilde{c}_t \circ (1 - \tilde{c}_t) \right).
\] (13)

Backpropagating, the network can learn which lags it wants to assign this importance to with (12), as we have a closed-form derivative effectively controlling which direction should this gate be updated to in order to better let through the input relevant at this specific time step. A similar interaction happens with gates \( u_t \) and \( \tilde{c}_t \), where \( u_t \) assigns importance (4) to the nonlinear outputs of \( \tilde{c}_t \) (3). Both are also optimized via backpropagation (11, 13).

We define \( W^a = (W_{ij})^T, i \leq h \) and \( W^x = (W_{ij})^T, i \geq h \) where \( h \) is the so-called hidden dimension of the cell, i.e. the parameter that defines the dimensionality of the gate output vectors and thus \( a_t \), since it consists of sums and Hadamard products of these activations.

The derivative w.r.t. the previous activation is then

\[
\frac{\partial L}{\partial a_{t-1}} = W_o^a \frac{\partial L}{\partial o_t} + W_u^a \frac{\partial L}{\partial u_t} + W_f^a \frac{\partial L}{\partial f_t} + W_{\tilde{c}_t}^a \frac{\partial L}{\partial \tilde{c}_t}
\]

and w.r.t. the input at time \( t, x_t \)

\[
\frac{\partial L}{\partial x_t} = W_o^x \frac{\partial L}{\partial o_t} + W_u^x \frac{\partial L}{\partial u_t} + W_f^x \frac{\partial L}{\partial f_t} + W_{\tilde{c}_t}^x \frac{\partial L}{\partial \tilde{c}_t}.
\]

### 2.3.3 Suitability to market data

Backpropagation through time in an LSTM computes the aforementioned derivatives at each time step and then updates parameters according to an update rule. A decent intuition is that an LSTM learns to assign importance to information at specific lags and due to the interactions of the gates can then trap this information for use in later time steps. These properties of the architecture and its gradient flow make long-short term memory networks ideal for our use case regarding financial time series. It can process variable length inputs, is still relatively computationally efficient and can handle long-range dependencies by design.

The LSTM is then our choice of architecture for our empirical analysis, as it has all the properties we deem necessary. The same properties make it well suited for many other venues of financial research as well. There is, however, still some preparation to be done
with market data before it’s easy for the LSTM to efficiently and reliably process.

2.4 Characteristic issues with market data and methods of mitigation

We present two characteristics inherent in market data relevant to our analysis, stochasticity and range discrepancies, and explain data preparation techniques that can be utilized to ease processing. Additional methods that were not required in this paper but can nonetheless be useful for related problems can be found in appendix B.

2.4.1 Stochasticity

Market data is largely stochastic. This becomes an issue with a model that deterministically optimizes according to an input-output-mapping, where an ex-post result of a stochastic process can be learned. This is a form of overfitting, and can be mitigated by a technique called regularization. Regularization in neural networks aims to increase out-of-sample performance (generalization) of a model by reducing the effect of noise.

We use ridge regression, which effectively increases generalization by discouraging model complexity by penalizing the loss function. This is done by adding a regularization parameter to the loss function so that

\[ L(y, \hat{y}) = L(y, \hat{y})' + \frac{1}{2} \lambda \sum_{i=1}^{k} W_i^2, \]

where \( \lambda \) is a hyperparameter. This has the effect of subtracting an amount from a given weight relative to that weights magnitude,

\[ \frac{\partial L}{\partial W} = \frac{\partial L'}{\partial W} - \lambda W. \]

2.4.2 Range discrepancies

Ranges of values for market data can be very small (e.g. daily returns for an asset) or very large (e.g. daily turnover of the same asset). Standardizing variables with
\[ x_{i,t-1} = \frac{x_i - \bar{x}}{\sigma(x)} \]

mitigates this, and storing \( \bar{x} \) and \( \sigma(x) \) in memory allows the transformation to be easily reversed.

### 2.5 Pairs trading

Pairs trading is a market-neutral strategy based on the divergence and subsequent convergence of comoving assets (Gatev et al., 2006). The intuitive theory behind it is that for two securities with similar assets underlying their intrinsic value, divergence between the two prices is largely attributable to security-specific shocks that will eventually converge. Taking a long position on the “downwards-diverging” one and a short position on the opposite and unwinding on convergence has been found to yield significant excess returns in multiple papers, e.g. (Broussard and Vaihekoski, 2012).

The strategy then has two steps, for pre-execution one must define rules for stock selection. This forming of pairs must be done prior to any trading occurring in what is called the formation period. There are multiple approaches, a good overview of which can be found in (Krauss, 2016). In this paper a historically valid pair, UPM and Stora-Enso, is first picked and analyzed out of the original period. Rinne and Suominen (2016) presented the same pair but only extended their analysis through 2003. This is intended to validate our model architecture. We then move to a cointegration-based approach, which Huck and Afawubo (2015) found to be the most reliable method.

Pairs trading profitability has been attributed to multiple factors. Gatev et al. (2006) surmise there is an undiscovered risk factor, while Andrade et al. (2005) present the returns are compensation for providing liquidity. Engelberg et al. (2009) also show liquidity is relevant to the spread and the convergence. Based on these results we want our model to account for liquidity and use daily volume as a proxy.

Gatev et al. (2006) analyze through 2002, which is also the year Do and Faff (2012) find the strategy largely unprofitable after. We conduct our initial analysis from 2010 through 2013, where linear methods have long since started to wane in performance (Engelberg et al., 2009). We then see if the same effects are still utilizable today.

It should be noted that our final experiments do not explicitly define a pairs trading strategy. We instead predict the spread of two stocks in a pair chosen with a well-
documented formation method based on previously empirically accurate predictors before constructing a portfolio with slightly altered properties explained in section 3. Recurrent neural networks have been used in related applications before with positive results. Huck (2009) uses Elman networks for return prediction in the S&P 100 before constructing a decision matrix of anticipated spreads. Our approaches are very similar in that both are concerned with the predictability of the spread; we, however, form our pair prediction.

After forming the pair we use a 5-day horizon for our predictions, motivated by positive results in Jylhä, Rinne and Suominen (2014) and Rinne and Suominen (2016). The former also motivated our choice of pair in our initial analysis. We, however, extend considerably beyond the period analyzed in these papers with a more complex model.

Other machine learning methods have also been used to find abnormal returns with pairs trading. Krauss et al. (2017) find out-of-sample daily returns of up to 0.45% within the S&P 500 using an ensemble model (one component of which is a deep neural network). Ensemble model is an umbrella term for any machine learning model that actually consists of multiple models operating together to gain performance superior to any of the models alone.

2.6 Cointegrated augmented Dickey-Fuller test

The augmented Dickey-Fuller test is a statistical test to test whether a process has a unit root. The test is called cointegrated when performed on the residuals of an ordinary least squares regression of two variables. It has $H_0$: The process contains a unit root, so a p-value below a critical threshold allows us to reject the null hypothesis that the residuals of the regression are nonstationary, i.e. it tells us the two series are likely cointegrated. We will later utilize this test to find pairs from CRSP price data.

3 Empirical Analysis

Related to our analysis, we present four things of note. Firstly, many of the mechanics of LSTMs are poorly understood, and inability to fit a model, even with an existing underlying independent-dependent variable mapping, is fairly common and can be due to a number of factors. These factors include but are not limited to model architecture, hy-
perparameterization, initialization and training time. An example of a failed model can be found in the case in appendix E. Even when properly fitted, any models presented are likely to be slightly nonoptimal. Each model was trained for a set number of epochs with no intra-training tuning or picking the best model from a set number of previous epochs.

Secondly, an additional constraint in our strategy compared to traditional pairs trading is that at each time step the portfolio must be long or short, i.e. we do not hold closed pairs. This effectively restricts us to constructing five separate portfolios with the investing based on whether it predicts the pair should be held long or short for one week, and should contribute to higher return volatility. All pairs presented are, however, nonetheless zero-investment so all returns can be considered excess.

Thirdly, in addition to t-tests with $H_0 : r = 0$ where $r$ is the weekly return, a binomial cumulative distribution function $\text{BinCDF}$ is used to signal statistical significance. Our assumption is that for a long-short portfolio in a pair of cointegrated stocks the unconditional probability of that portfolio going up is 0.5. The assumption seems reasonable in our samples.

Finally, we would like to highlight that due to look-ahead bias no results in sections 3.1 and 3.2 can be considered statistically significant and should be considered preliminary tests for model engineering before conducting the actual statistical testing and obtaining the results in section 3.3.

### 3.1 Case UPM-Stora Enso

We conduct analysis with our target vector being the movement of the spread in one week (five time steps into the future). We use a stateful LSTM, which feeds the last activation and cell state of the previous training batch to the model.

Note that this first case aims to validate our model for generalization, i.e. our $H_0$ is that our model can have no significant predictive power over the spread temporally out-of-sample. We thus do not utilize separate validation and test sets. There is then inherent look-ahead bias in the returns. The same pair was also explored by Rinne and Suominen (2016) with positive results in an earlier period.
3.1.1 Data acquisition

Training set. We take historical data from Nasdaq Historical Prices 1997-01-01 to 2010-01-01 for UPM and Stora. We then take the difference in closing price $P_{UPM} - P_{Stora}$ to construct the returns of our long pair, and take the difference of lag 5.

For our input variables, we use closing price and daily volume for both stocks. Closing price is included for autocorrelations, volume as a measure of liquidity. All variables are standardized (according to section 2.4.2) prior to input. The training standardization parameters are used throughout the test set as well.

Validation-testing set. We take the same series from 2010-01-01 to 2013-12-31. We use this set as both the validation set and the test set, since this pair is an exploration into the validity of our model.

3.1.2 Methodology

The same stateful LSTM model will be used to predict throughout the whole period with a fixed number of epochs. If the model fails to sufficiently minimize training loss, generalize to the validation set or fails to converge, we abort training and manually tune our hyperparameters. We then retrain until we find a set of hyperparameters that allow our model to fit the data.

3.1.3 Results

Portfolio performance. For our test set, the model succeeds in predicting the movement of the long-short pair 646 times out of a total of 998 ($BinCDF(646, 998, 0.5) = 1.11 \times 10^{-16}$), achieving an average weekly (log) return of 2.38% across the five portfolios with a volatility of 7.6%. Since there is clearly a set of circumstances where the model can make statistically significant predictions temporally out-of-sample, we reject the null hypothesis. The returns themselves are however not statistically significant.

Figure 2 truncates the last year to highlight several significant drops in the value the portfolios in 2012, which might help explain poor performance in that period apparent in Table 1. Similarly formatted tables specific to the five portfolios can be found in appendix C. The relatively poor performance in 2012 is peculiar, and can be attributed mostly to a
few large drops in the best performing portfolios during that year. This could be due to a number of factors, e.g. an uncharacteristic data period, a period of uncertainty increasing the effect of stochastic factors or just a less-than-optimally trained or designed model.

<table>
<thead>
<tr>
<th>Period</th>
<th>Sharpe</th>
<th>$\sigma$</th>
<th>$r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>0.313</td>
<td>0.076</td>
<td>0.0238</td>
</tr>
<tr>
<td>2010</td>
<td>0.533</td>
<td>0.064</td>
<td>0.0341</td>
</tr>
<tr>
<td>2011</td>
<td>0.417</td>
<td>0.066</td>
<td>0.0275</td>
</tr>
<tr>
<td>2012</td>
<td>0.128</td>
<td>0.078</td>
<td>0.01</td>
</tr>
<tr>
<td>2013</td>
<td>0.261</td>
<td>0.092</td>
<td>0.024</td>
</tr>
</tbody>
</table>

*Table 1: Weekly returns and volatilities for different periods.*
Prediction vector period length distribution and model behaviour. Figure 3 shows the length of consecutive time steps of long or short positions in the complete prediction vector (that is pre-portfolio splitting). The model thus predicted that the 1-week spread changes direction on average once every 7.61 days.

In Figure 4 we can see the model captures effects similar, although seemingly more complex compared to a pairs trading strategy. The graph shows both prices from 2010 through 2013 normalized by division with their mean as well as a line depicting how many of our five portfolios were open at that point in time.
3.2 CRSP 2010-2013

Considering the success of UPM-Stora, we extend the analysis to a larger universe of stocks within the same period.

$H_0$: UPM-Stora was a pair with particular properties and the results are not reproducible in other pairs formed with a historically valid metric (cointegration).

3.2.1 Data acquisition

We take the CRSP universe of stocks. Our picking algorithm is as follows:

1. Pick two stocks at random.
2. If the two stocks are the same, return to 1.
3. Index both series with the intersection of their date indices (SQL-style inner join).
4. If the length of the joined series before or after our date cutoff is less than a predetermined value, return to 1.
5. If the p-value of a cointegrated augmented Dickey-Fuller test is larger than a predetermined value (pre-cutoff), return to 1.
6. Yield the pair.

If the function yields a pair, we try to fit a model to it. If that model is successful, we save the model. If not, we form a new pair. While this might seem like cherry-picking, it should be noted that even in the presence of a mapping there is no guarantee an LSTM will fit to it due to initialization stochasticity. Further, this analysis is a continuation of UPM-Stora in that it only aims to find out whether our network architecture can feasibly fit to out-of-sample data.

Anecdotally, our algorithm fit one successful model for around three pairs returned by our picking function, and e.g. first fitting the UPM-Stora model took five to ten training runs with the final hyperparameters. We might thus be effectively throwing away perfectly good pairs when only training once.

**Training set.** From a given pair we take the inner join of the two until 2010-01-01, provided there are at least 1500 observations (~6 years). This also serves as our (considerably long) formation period for the pair. Preparation is identical to UPM-Stora.
Validation-testing set. We take the same pair until 2013-12-31 provided there are at least 500 examples (~2 years). Any returns are thus not statistically significant due to look-ahead bias.

3.2.2 Methodology

We train identically with UPM-Stora for each pair, saving a model if we observe testing accuracy above a threshold after the last epoch. If not, we disregard that pair. We thus expect to see superficially highly significant results.

3.2.3 Results

<table>
<thead>
<tr>
<th>Pair</th>
<th>Sharpe</th>
<th>$\sigma$</th>
<th>$r$</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACY-PVFC</td>
<td>0.038</td>
<td>0.448</td>
<td>0.0169</td>
<td>4.99</td>
<td>3.535E-07</td>
</tr>
<tr>
<td>BKE-YDNT</td>
<td>0.433</td>
<td>0.209</td>
<td>0.0904</td>
<td>12.94</td>
<td>4.667E-35</td>
</tr>
<tr>
<td>CAC-SRI</td>
<td>0.462</td>
<td>0.056</td>
<td>0.0260</td>
<td>14.93</td>
<td>6.327E-46</td>
</tr>
<tr>
<td>GNCMA-NCU</td>
<td>0.561</td>
<td>0.100</td>
<td>0.0562</td>
<td>19.07</td>
<td>1.174E-69</td>
</tr>
<tr>
<td>MLP-GRC</td>
<td>0.221</td>
<td>0.145</td>
<td>0.0320</td>
<td>8.51</td>
<td>3.263E-17</td>
</tr>
<tr>
<td>MTR-ALXN</td>
<td>0.181</td>
<td>0.141</td>
<td>0.0256</td>
<td>6.59</td>
<td>3.493E-11</td>
</tr>
<tr>
<td>MTR-LABL</td>
<td>0.260</td>
<td>0.071</td>
<td>0.0185</td>
<td>9.01</td>
<td>5.012E-19</td>
</tr>
<tr>
<td>PMFG-AAME</td>
<td>0.255</td>
<td>0.112</td>
<td>0.0287</td>
<td>9.06</td>
<td>3.276E-19</td>
</tr>
<tr>
<td>ROCM-HRL</td>
<td>0.188</td>
<td>0.386</td>
<td>0.0726</td>
<td>6.92</td>
<td>3.984E-12</td>
</tr>
<tr>
<td>SBR-THFF</td>
<td>0.605</td>
<td>0.084</td>
<td>0.0509</td>
<td>20.21</td>
<td>1.188E-76</td>
</tr>
</tbody>
</table>

*Table 2: Pair-specific statistics for CRSP 2010-2013.*

The p-values of the t-tests in table 2 are considerably small, which is to be expected, since we essentially engineered them to be. While the $n$ is small and we recognize our look-ahead bias means these returns are not statistically significant, the positive Sharpes and excess returns achieved by a low enough number of training runs are enough to continue our analysis. We consider the results of UPM-Stora reproducible in a larger subset of the stock space and reject the null hypothesis.
3.3 CRSP 2018-2019

We use the same picking algorithm for data acquisition as our 2010-2013 analysis with the exception that instead of utilizing training loss we create a separate validation set. Here we have $H_0$: Our previous tests were affected by look-ahead bias and excess returns cannot be generated in its absence ceteris paribus.

Training set. We take any observations prior to 2017, then standardize. Both the period and the preparation are then identical to previous cases. Pair formation is identical to CRSP 2010-2013.

Validation set. We validate the model 01-01-2017 to 31-12-2017, evaluating the accuracy of the model during this period. We consider any model with a validation accuracy of over 65% to be successful.

Testing set. Results presented are from 01-01-2018 until the latest observation on the specific pair available from CRSP quarterly on 2019-05-25, which for a typical pair is 2019-03-29. This is a testing set that has never been seen by the model.

3.3.1 Methodology

Our prediction methodology is identical to the previous tests with the exception of validating on a separate dataset. We also construct a portfolio representative of both survivors and non-survivors. We consider a pair bankrupt over period if there is need to completely eliminate a risk-free security held for collateral, i.e. we unwind the pair when the current value of the 1:1 long-short hits 0. We then pay any occurred 1-period losses out of the portfolio value and bankrupt pairs are then held in the $\frac{1}{n}$ portfolio with capital 0. This eliminates runaway models as we have little visibility into the inner decision making of the networks. No rebalancing is made in the period.

3.3.2 Results

Volatility is calculated across the five portfolios while $r$ is the geometric weekly return of holding all portfolios over the period.
<table>
<thead>
<tr>
<th>Pair</th>
<th>Sharpe</th>
<th>$\sigma$</th>
<th>$r$</th>
<th>t-statistic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOR-KOSS</td>
<td>0.473</td>
<td>0.154</td>
<td>0.0727</td>
<td>8.50</td>
<td>5.627E-16</td>
</tr>
<tr>
<td>ADRE-FLO</td>
<td>0.685</td>
<td>0.065</td>
<td>0.0443</td>
<td>11.33</td>
<td>4.639E-25</td>
</tr>
<tr>
<td>AMSWA-PGC</td>
<td>0.448</td>
<td>0.064</td>
<td>0.0285</td>
<td>7.82</td>
<td>5.528E-14</td>
</tr>
<tr>
<td>BOSC-NBN</td>
<td>0.668</td>
<td>0.047</td>
<td>0.0315</td>
<td>11.32</td>
<td>4.951E-25</td>
</tr>
<tr>
<td>EBTC-TKC</td>
<td>-0.046</td>
<td>0.084</td>
<td>-0.0039</td>
<td>-0.13</td>
<td>4.496E-01</td>
</tr>
<tr>
<td>EIS-HBNC</td>
<td>0.256</td>
<td>0.106</td>
<td>0.0272</td>
<td>4.99</td>
<td>5.271E-07</td>
</tr>
<tr>
<td>EXI-SQM</td>
<td>0.340</td>
<td>0.015</td>
<td>0.0051</td>
<td>5.79</td>
<td>9.564E-09</td>
</tr>
<tr>
<td>FDD-ETY</td>
<td>0.568</td>
<td>0.028</td>
<td>0.0157</td>
<td>9.79</td>
<td>5.642E-20</td>
</tr>
<tr>
<td>FISI-MFL</td>
<td>0.329</td>
<td>0.186</td>
<td>0.0613</td>
<td>6.39</td>
<td>3.451E-10</td>
</tr>
<tr>
<td>FLXS-INDB</td>
<td>0.003</td>
<td>0.433</td>
<td>0.0014</td>
<td>3.52</td>
<td>2.548E-04</td>
</tr>
<tr>
<td>LBIX-NKG</td>
<td>0.207</td>
<td>0.029</td>
<td>0.0060</td>
<td>2.65</td>
<td>4.522E-03</td>
</tr>
<tr>
<td>MAT-FELE</td>
<td>0.661</td>
<td>0.087</td>
<td>0.0574</td>
<td>10.90</td>
<td>1.347E-23</td>
</tr>
<tr>
<td>MCN-CUR</td>
<td>0.547</td>
<td>0.048</td>
<td>0.0260</td>
<td>9.42</td>
<td>8.537E-19</td>
</tr>
<tr>
<td>MOG-FXF</td>
<td>0.616</td>
<td>0.094</td>
<td>0.0577</td>
<td>10.81</td>
<td>2.626E-23</td>
</tr>
<tr>
<td>NUM-APT</td>
<td>0.313</td>
<td>0.018</td>
<td>0.0058</td>
<td>5.37</td>
<td>8.379E-08</td>
</tr>
<tr>
<td>PAI-CRESY</td>
<td>0.769</td>
<td>0.105</td>
<td>0.0810</td>
<td>13.53</td>
<td>9.016E-33</td>
</tr>
<tr>
<td>PCYO-OVBC</td>
<td>0.422</td>
<td>0.078</td>
<td>0.0329</td>
<td>7.52</td>
<td>3.726E-13</td>
</tr>
<tr>
<td>PDM-FOE</td>
<td>0.550</td>
<td>0.065</td>
<td>0.0358</td>
<td>9.84</td>
<td>4.083E-20</td>
</tr>
<tr>
<td>PGF-KEP</td>
<td>0.701</td>
<td>0.108</td>
<td>0.0755</td>
<td>12.37</td>
<td>1.155E-28</td>
</tr>
<tr>
<td>RHI-BMTC</td>
<td>0.192</td>
<td>0.104</td>
<td>0.0200</td>
<td>3.56</td>
<td>2.140E-04</td>
</tr>
<tr>
<td>SGC-FIT</td>
<td>0.458</td>
<td>0.046</td>
<td>0.0212</td>
<td>7.80</td>
<td>5.960E-14</td>
</tr>
<tr>
<td>SON-BIS</td>
<td>0.668</td>
<td>0.093</td>
<td>0.0621</td>
<td>11.01</td>
<td>5.638E-24</td>
</tr>
<tr>
<td>TJX-PDM</td>
<td>0.128</td>
<td>0.526</td>
<td>0.0671</td>
<td>2.93</td>
<td>1.845E-03</td>
</tr>
<tr>
<td>VLGEA-YUM</td>
<td>0.494</td>
<td>0.085</td>
<td>0.0419</td>
<td>8.83</td>
<td>5.715E-17</td>
</tr>
</tbody>
</table>

Bankrupt pairs

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CHA-BJZ</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EEI-ATAX</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IHT-OC</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>UXI-ALTR</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>USAK-OPOF</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Portfolio (1/n) | 0.505  | 0.104  | 0.0526 |

*Table 3: Pair-specific and portfolio statistics for CRSP 2018-2019.*

Table 3 shows statistically significant positive returns for the vast majority of portfolio
components and form an $\frac{1}{n}$ portfolio with a weekly Sharpe ratio of 0.505. We can see that while bankruptcy risk was potentially heavily mitigated by look-ahead bias in previous results, the returns remain significant after this bias is eliminated. We consider the results replicable in the absence of look-ahead bias and reject the null hypothesis.

Additionally, in this larger sample with a separate training set, the prediction vector period length distribution compared to UPM-Stora was smaller in mean absolute value and larger in volatility with a mean of 5.76 and a standard deviation of 10.4.

4 Discussion

Our results indicate that nonlinear models can have significant predictive power over the spread of cointegrated stocks. The effectiveness as well as the long formation and training periods of both the 2010-2013 and the 2018-2019 tests indicate some effects persistent throughout the sample are still observable, and pairs trading can produce statistically significant excess returns as recently as early 2019.

Some of our stocks in this latter period ended in an artificial bankruptcy. On average the spread prediction on bankrupt pairs was also highly significant but had an early period of low performance. While the trading strategy could have been held through these periods, we felt the value crossing below zero to be a valid control for a runaway model since there is no real visibility to how an LSTM will react to a given situation. We thus implemented it as our “big red button” to stop trading.

The same lack of visibility motivating our threshold for unwinding a pair unfortunately also contributes to our poor understanding of how these excess returns actually came to be. The black box analogy regarding neural networks is, sadly, in many regards still rather accurate. Most we can conclude is that there are statistically significant profitable relationships between the historical prices and volumes of cointegrated stocks and their future spread. While linear methods can no longer capture these relationships, our results suggest they still exist and can be exploited by more complex models.

There are regrettably few conclusions we can draw on what characteristics of movement in the input variables these returns were driven by. This is due to the fact that contrary to a traditional regression where it is simple to see the effects the independent variables have on the dependent, the input-output mapping of an LSTM is inherently very complex. This is exemplified by table 6 in appendix D, where we correlated one-
week aggregate volume of the two stocks in the pair with their portfolio return. Similar results were found with other weightings and windows.

The results seem somewhat haphazard, and suggest there is no simple linear relationship and that some factors related to liquidity might even exhibit pair-specific variation, which our network can then exploit. Note that too many conclusions shouldn’t be drawn, as the p-value of the Pearson correlation is known to misbehave on sample sizes as small as ours. Nonetheless, our tests do not find a similar relationship between short-term liquidity shocks and utilizing the spread for returns as previous literature and suggest our networks are utilizing more complex, nonlinear relationships.

While we can’t easily extract information on the internal mappings of the network, we can draw try to analyze and remap its post-activation behavior. One avenue for this is seeing how often the model likes to flip the pair. The distribution of period lengths in the prediction vector suggests strongly performing portfolios could be constructed by relaxing our constraint of “all-or-nothing”, scaling the weight according to a metric of model stability.

This would be implemented by requiring the model to be more “certain”, i.e. further away from the decision plane, in order take the portfolio inverse to the previously held one. A simple 1:1 long-short was selected and used throughout to avoid personal look-ahead bias, but two possible venues of potentially justified weighting adjustment are recognized. Firstly, simple weighing according to the model confidence of prediction \( \hat{y}_{t+n} \) at \( t \) and secondly a switching penalty where longer holding periods would be preferred. This would have the added benefit of increasing the robustness of the returns \( w.r.t. \) trading costs, which are currently completely absent.

Another analysis we can conduct based on this post-activation behavior is whether there seems to be a possible latent risk factor as surmised by Gatev et al. (2006). We do not find any evidence supporting this, as the mean p-value for a Pearson correlation between the prediction vectors of two given pairs is \(~0.3\), and within the matrix we only have two correlations significant at the 1% level. The returns tell a similar story, and the amount of pairs open on a given day also mostly resembles a stochastic process we could not extract meaningful information from; a graph is nonetheless presented in appendix D.

Also in appendix D can be found the market capitalizations for each stock in a given pair. The model seems to have (with a relatively low \( n \)) no regard to the market capitalization within the pairs it manages to fit itself into. If the models would only seek
out apparent statistical arbitrage in low-cap stocks, all the returns would be virtual. This is not the case, however, and we consider the vast majority of the trades in the period executable.

As a final note, dynamic training methods should be utilized when creating these models. A quick non-documented test concluded that the specific model used for predicting UPM-Stora did not generalize to other stocks, which could be due to a variety of different data characteristics. We thus trained separate models for each pair. Further analysis would be required to extract any meaningful information out of the similarities or differences of these separate models, but could yield interesting results. A venue is surmised in section 5.4.1.

5 Conclusions

5.1 Research summary

We find statistically significant excess returns predicting the spread of two cointegrated stocks as recently as 2018-2019, a period previously unexplored, with a model architecture that has seen very little use in financial literature. Our results also indicate constructing these successful LSTMs for financial time series might not require massive amounts of data or computational power. Many previous implementations successful in gaining insight (e.g. Sirignano and Cont, 2018) utilize considerable amounts of both. We have also contributed to bridging the gap between financial analysis and machine learning by recognizing and mitigating issues with traditional market data, as well as presenting an established model well suited for many areas of market prediction. However, results are still (as with all neural networks) often heavily dependent on both model design and very specific data preparation.

5.2 Practical implications

Apart from the significant returns from trades we concluded to be on average highly executable, the general dimensionality of the pair-specific model architecture is minimal, meaning a prediction can be made in very little time. If similar market performance can be found with higher frequency data, training (and utilizing) this type of architecture could be done online (with stochastic gradient descent) for high-frequency training.
High-frequency pairs trading has also been shown to be feasible in the past (Bowen et al., 2010). All models used were trained on inexpensive consumer grade hardware.

5.3 Limitations of the study

The study in no way considers trading costs, and the CRSP stock space is almost completely unfiltered. This casts doubt on whether a small subset of the pairs in the 2018-2019 portfolio are tradable, although most of them without a doubt are. Further, while our pair formation is technically survivorship bias-robust by design as it has no information on whether a stock still exists out of the training set, a larger sample size would be needed to rigorously show related effects.

From the mathematical and technical perspective, LSTMs are still poorly understood. This limited our study in two ways. Firstly, there was very limited visibility into the decision making of the model and thus the factors it prioritized when predicting the spread. Secondly, post-training performance of a model is largely stochastic itself, since for given hyperparameters and initialization finding a solution close to a global minimum (maximum) is not guaranteed for any non-convex (non-concave) mapping. Whether our rate of successful post-training models was negatively affected by this or other factors is unclear. The same hyperparameters were also used throughout the stock space when there is no particular reason to believe they are optimal for each pair. While this method was unlikely to increase performance, rather the opposite, it did potentially limit our sample size due to time wasted training unused models. This could have been mitigated with dynamic hyperparameterization, or increased processing power or time.

5.4 Further research

Three general areas of possible future research are presented: sample size, time dependency and model architecture. While portfolio performance is considered, the main area of interest is the decision making of the model and which aspects of it are constant versus variable throughout time and stock space. This could yield important insight to market factors.
5.4.1 Sample size

The most obvious venue of further research is generalization of the model to a wider space of stocks. A larger number of pairs selected by either traditional methods or another machine learning model coupled with automated (e.g. grid search) hyperparameterization seems a reasonable method. Guidelines can be found in (Greff et al., 2017), which finds the network size and learning rate to be the most generally tunable hyperparameters. This venue would also potentially allow for the design and training of a general “pairs-network”, analysis of which would be a much more efficient use of time than trying to draw conclusions from multiple separate single-pair models.

A temporally larger sample, on the other hand, would allow us to reliably test whether we find evidence on pairs trading returns being applicable to liquidity in a more complex way than previously thought. A possible method for liquidity could be to compare two models, only one of which has been given volume data, and evaluating performance during periods of varying liquidity. We do not feel confident in this with the current length of the sample, and it is thus not presented.

5.4.2 Time dependency

There is also insight to be gained researching the validity of the model over time in a more rigorous fashion, as well as further exploring the differences in the weight structure of optimized models over time. This could yield insight into the market adoption of different types of predictors.

Additionally, our formation and training period was very long compared to our investment horizon. This was to enable capturing more rigorous effects, but has been shown (Huck and Afawubo, 2015) to be highly tunable regardless of the selection method.

5.4.3 Model architecture

The current models simply take in consecutive trading days. Taking times of week or bank holidays into account via data preparation could help the model find related effects.

Another interesting topic could be valuating the model performance after modifying the target variable to a ternary vector explicitly describing the time steps to optimally have a pair long, short or closed. This could significantly lower portfolio volatility.
Finally, the large return volatilities in our portfolios and the poor visibility into the investment decisions made by the model raises questions about trust. There is little predictability in whether the model will decide to bankrupt itself at $t + 1$, and any mechanisms additional to our artificial bankruptcy to control this are areas of interest.

6 References


Glorot, X., Bengio, Y. (2010). Understanding the difficulty of training deep feedfor-


7 Appendices

7.1 Appendix A: LSTM implementation

Activation functions:

In [ ]: def sigmoid(Z):
             #Also returns original to help with backprop
             return 1/(1+np.exp(-Z)), Z

def d_sigmoid(cache):
    s, _ = sigmoid(cache)
    return s * (1-s)

def tanh(Z):
    #Also returns original to help with backprop
    A, _ = sigmoid(Z * 2)
    return A * 2 - 1, Z

def d_tanh(cache):
    t, _ = tanh(cache)
    return (1 - t**2)

Implementation of forward propagation:
In [ ]: def forward(self, x, a_prev, c_prev):
    """Cell forward.
    Args:
        x: Input data.
        a_prev: Activation at t-1.
        c_prev: Cell state at t-1.
    Returns:
        state: Dictionary of states of the different activations at t.
        cache: Dictionary of gate activation function inputs.
    """
    a_prev = a_prev if a_prev is not None
    else np.zeros((self.hidden_dim, x.shape[0]))
    c_prev = c_prev if c_prev is not None
    else np.zeros((self.hidden_dim, x.shape[0]))

    state = {}
    state['c_in'] = c_prev
    state['z'] = np.vstack((a_prev, x.T))

    cache = {}
    for k, func in selffuncs.items():
        state[k], cache[k] = func['a'](  
            np.dot(self.params[k]['w'], state['z']) + self.params[k]['b']  
        )

    state['c_out'] = state['f'] * c_prev + state['u'] * state['c']
    state['a_out'] = state['o'] * tanh(state['c_out'])[0]

    return state, cache

Implementation of model forward propagation:

In [ ]: def forward(self, x, y, a_prev=None, c_prev=None):
    """Forward prop.
    Args:
    """
x: Input array to network.
    Shape (time_steps, batch_size, x_dim).
y: Post-activation target variables.
    Shape (time_steps, batch_size, output_dim).
a_prev: Previous cell (pre-post-cell) activation.
    If None, will be initialized in cell.
c_prev: Previous cell state.
    If None, will be initialized in cell.

Returns:
    states: List of dictionaries of cell states.
    caches: List of dictionaries of cell activation function inputs.
    preds: Network predictions.
    targets: Unchanged target variable (pass through)

states, caches, preds = [], [], []
x = [x[t,:,:] for t in range(x.shape[0])]
y = [y[t,:,:] for t in range(y.shape[0])]
for xt in x:
    state, cache = self.cell.forward(xt, a_prev, c_prev)
    states.append(state)
    caches.append(cache)
    pred = self.activation.forward(state['a_out'])
    preds.append(pred.T)
a_prev, c_prev = state['a_out'], state['c_out']
return states, caches, np.stack(preds), np.stack(y)

Implementation of backpropagation of single cell:

In [ ]: def backward(self, state, da_next, dc_next):
    """Cell backward.
    Inputs:
        state: cell state at t
        da_next: activation gradient of cell t
            (gradient w.r.t activation input to t+1)
        dc_next: cell gradient of cell t
(gradient w.r.t cell input to t+1)

Returns:

da_in: activation gradient of cell t-1
  (gradient w.r.t activation input to t)

dc_in: cell gradient of cell t-1
  (gradient w.r.t cell input to t)

grads: dictionary of gate gradients for updating params

"""

dc_out = state['o'] * da_next * d_tanh(state['c_out']) + dc_next
grads = self.init_grads()

d = {}

d['c'] = (1 - state['c']**2) * state['u'] * dc_out

d['u'] = state['u'] * (1 - state['u']) * state['c'] * dc_out

d['o'] = (state['o'] * (1 - state['o']))
  * tanh(state['c_out'])[0] * da_next

d['f'] = (state['f'] * (1 - state['f']))
  * state['c_in'] * dc_out

da_in = np.zeros_like(da_next)

for gate in ['c', 'u', 'o', 'f']:
    da_in += np.dot(self.params[gate]['w'].T[:self.hidden_dim,:], d[gate])

    grads[gate]['b'] = np.sum(d[gate], axis=1, keepdims=True)

    grads[gate]['w'] = np.dot(d[gate], state['z'].T)

dc_in = dc_out * state['f']

return da_in, dc_in, grads

Implementation of network backpropagation:

In [ ]: def backward(self, states, caches, preds, targets):
  """Network backprop.

  Updates parameters of cell and activation function.

  Args:

    states: list of dictionaries containing states from LSTM_unit
caches: caches, not needed - prime for a refactor
preds: list of predictions (\(y_{\text{hat}}\)) of shape
(output_dim, batch_size)
targets: list of labels with the same shape as preds
grad_check: if True, cell will store pre-clipped grads
in cell.grads on param update

Returns:
preds: network predictions, unchanged (pass-through)
targets: target variable, unchanged (pass-through)

```python
d_loss = self.dloss(preds, targets)
das = [self.activation.backward(d_loss[t]).T for t in range(d_loss.shape[0])]
da_next = np.zeros_like((das[0]))
dc_next = np.zeros_like((states[0]['c_out']))
grads = self.cell.init_grads()
for state, da in zip(reversed(states), reversed(das)):
da_next += da

da_next, dc_next, grad_adds = self.cell.backward(state, da_next, dc_next)
for gate in ['c', 'u', 'o', 'f']:
grads[gate]['w'] += grad_adds[gate]['w']
grads[gate]['b'] += grad_adds[gate]['b']
self.cell.update_params(grads, self.grad_clip)
sel.activation.update_params()
return preds, targets
```

After calculating \(\sum_{i=1}^{T} \frac{\partial L}{\partial x_{i}}\), we can update the parameters of the network.

```
In []: def update_params(self, grads, clip=None):
if self.grad_check:
    self.grads = grads
for gate in ['c', 'u', 'o', 'f']:
    if clip is not None:
        grads[gate]['w'] = np.clip(grads[gate]['w'], -clip, clip)
grads[gate]['b'] = np.clip(grads[gate]['b'], -clip, clip)
self.params[gate]['w'] *= self.learning_rate
self.params[gate]['b'] *= self.learning_rate
```
Loss functions:

```python
In [ ]: class L2_loss:
    @classmethod
    def loss(self, y_hat, y):
        return (y_hat - y) ** 2

    @classmethod
    def dloss(self, y_hat, y):
        return (y_hat - y) * 2

class CrossEntropyLoss:
    @classmethod
    def loss(self, y_hat, y):
        return -np.mean(Y*np.log(AL) + (1-Y)*np.log(1-AL))

    @classmethod
    def dloss(self, y_hat, y):
        return -(np.divide(y, y_hat) - np.divide(1-y, 1-y_hat))
```

### 7.2 Appendix B: Other financial problems for neural networks

**Softmax and portfolio weights.** Convenitely, the softmax function

\[ \sigma(x_1, x_2, ..., x_n)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \]

found in many machine learning libraries maps values to interval \((0, 1)\) with the components summing up to 1. This makes them interpretable as the weights in a long portfolio, and facilitates optimizing portfolio weights w.r.t. an arbitrary function trivial. Taking

\[ y_i = x_i - \frac{1}{n}, \{i \in \mathbb{Z} | i \leq n\} \]

allows us to create zero-investment portfolios.
Up-or-down-mapping. A simple alternative to our approach for mapping a target variable to whether a given instrument will go up or down after $m$ periods from $t$ is to define

$$y_t = \begin{cases} 
1 & \text{if } y_{t+m} \geq y_t \\
0 & \text{otherwise.}
\end{cases}$$

Data preparation example. Presented for a solution not utilizing an intermediary function, i.e. $\hat{y}_t = a_t$. To squash a given price series $p$ to the range of the hyperbolic tangent one can define

$$r_i = \begin{cases} 
\frac{p_i - p_{i-1}}{p_{i-1}} - 1 & \text{if } p_i \leq 2p_{i-1} \\
1 & \text{otherwise.}
\end{cases}$$

This has only the trivial shortcoming of eliminating single-period returns of $> 1$ from the range.

The derivative for the L2 loss is

$$2(y - \hat{y}_t).$$

Normalizing our target data between -1 and 1 will allow us to use $a_t$ as our prediction ($o_t$ is an output of $\sigma$ which produces values between 0 and 1). Thus

$$\frac{\partial \hat{y}}{\partial a_t} = 1$$

and

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_t} = \frac{\partial L}{\partial \hat{y}} = 2(y - \hat{y}).$$

The gate derivatives then simplify to

$$\frac{\partial L}{\partial u_t} = 2(y - \hat{y}_t) \circ o_t \circ (1 - o_t)$$
\[ \frac{\partial L}{\partial u_t} = \frac{\partial L}{\partial c_t} \circ \hat{c}_t \circ u_t \circ (1 - u_t) \]

\[ \frac{\partial L}{\partial f_t} = \frac{\partial L}{\partial c_t} \circ c_{t-1} \circ f_t \circ (1 - f_t) \]

\[ \frac{\partial L}{\partial \tilde{c}_t} = \frac{\partial L}{\partial c_t} \circ u_t \circ \tilde{c}_t \circ (1 - \tilde{c}_t) \]

where

\[ \frac{\partial L}{\partial c_t} = 2(y - \hat{y}_t) \circ o_t \circ (1 - tanh(c_t))^2 + \frac{\partial L}{\partial c_{t+1}}. \]

### 7.3 Appendix C: Portfolio-specific returns and volatilities for UPM-Stora

Portfolios are sorted in descending mean return order.

<table>
<thead>
<tr>
<th>Portfolio</th>
<th>Period</th>
<th>Sharpe</th>
<th>σ</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Total</td>
<td>0.312</td>
<td>0.094</td>
<td>0.0293</td>
</tr>
<tr>
<td>2010</td>
<td>0.48</td>
<td>0.084</td>
<td>0.0403</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>0.43</td>
<td>0.088</td>
<td>0.0378</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>0.14</td>
<td>0.096</td>
<td>0.0134</td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>0.273</td>
<td>0.098</td>
<td>0.0268</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Total</td>
<td>0.253</td>
<td>0.105</td>
<td>0.0266</td>
</tr>
<tr>
<td>2010</td>
<td>0.44</td>
<td>0.087</td>
<td>0.0383</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>0.31</td>
<td>0.087</td>
<td>0.027</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>0.034</td>
<td>0.128</td>
<td>0.0044</td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>0.276</td>
<td>0.102</td>
<td>0.0282</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Total</td>
<td>0.243</td>
<td>0.099</td>
<td>0.0241</td>
</tr>
<tr>
<td>2010</td>
<td>0.473</td>
<td>0.085</td>
<td>0.0402</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>0.152</td>
<td>0.111</td>
<td>0.0169</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>0.109</td>
<td>0.11</td>
<td>0.012</td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>0.261</td>
<td>0.075</td>
<td>0.0196</td>
<td></td>
</tr>
<tr>
<td>Portfolio</td>
<td>Period</td>
<td>Sharpe</td>
<td>σ</td>
<td>r</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>--------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>4</td>
<td>Total</td>
<td>0.168</td>
<td>0.114</td>
<td>0.0191</td>
</tr>
<tr>
<td>2010</td>
<td>0.291</td>
<td>0.091</td>
<td>0.0265</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>0.2</td>
<td>0.097</td>
<td>0.0194</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>0.016</td>
<td>0.117</td>
<td>0.0019</td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>0.091</td>
<td>0.127</td>
<td>0.0115</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Total</td>
<td>0.143</td>
<td>0.115</td>
<td>0.0165</td>
</tr>
<tr>
<td>2010</td>
<td>0.111</td>
<td>0.122</td>
<td>0.0135</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>0.164</td>
<td>0.108</td>
<td>0.0177</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>0.004</td>
<td>0.124</td>
<td>0.0005</td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>0.239</td>
<td>0.096</td>
<td>0.0229</td>
<td></td>
</tr>
</tbody>
</table>

*Table 4: Portfolio-specific statistics for the five portfolios generated in UPM-Stora.*

### 7.4 Appendix D: Additional tables and figure for CRSP 2010-2019

A dashed line in table 5 indicates the instrument does not have a meaningful market capitalization or it could not be found.

<table>
<thead>
<tr>
<th>Pair</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOR-KOSS</td>
<td>460,12M</td>
<td>15,18M</td>
</tr>
<tr>
<td>ADRE-FLO</td>
<td>17,69M</td>
<td>4,71B</td>
</tr>
<tr>
<td>AMSWA-PGC</td>
<td>369,07M</td>
<td>537,59M</td>
</tr>
<tr>
<td>BOSC-NBN</td>
<td>9,64M</td>
<td>193,04M</td>
</tr>
<tr>
<td>EBTC-TKC</td>
<td>344,77M</td>
<td>10,93B</td>
</tr>
<tr>
<td>EIS-HBNC</td>
<td>-</td>
<td>716,34M</td>
</tr>
<tr>
<td>EXI-SQM</td>
<td>204,85M</td>
<td>3,58B</td>
</tr>
<tr>
<td>FDD-ETY</td>
<td>-</td>
<td>1,70B</td>
</tr>
<tr>
<td>FISI-MFL</td>
<td>432,85M</td>
<td>493,78M</td>
</tr>
<tr>
<td>FLXS-INDB</td>
<td>138,86M</td>
<td>2,50B</td>
</tr>
<tr>
<td>LBIX-NKG</td>
<td>12.43M</td>
<td>129,69M</td>
</tr>
<tr>
<td>MAT-FELE</td>
<td>3,70B</td>
<td>2,05B</td>
</tr>
<tr>
<td>MCN-CUR</td>
<td>180,42M</td>
<td>9,41M</td>
</tr>
<tr>
<td>MOG-FXF</td>
<td>2,96B</td>
<td>-</td>
</tr>
<tr>
<td>Pair</td>
<td>left</td>
<td>right</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>NUM-APT</td>
<td>276,17M</td>
<td>5,74B</td>
</tr>
<tr>
<td>PAI-CRESY</td>
<td>137,01M</td>
<td>5,05B</td>
</tr>
<tr>
<td>PCYO-OVBC</td>
<td>230,07M</td>
<td>178,65M</td>
</tr>
<tr>
<td>PDM-FOE</td>
<td>2,54B</td>
<td>1,16B</td>
</tr>
<tr>
<td>PGF-KEP</td>
<td>22,96M</td>
<td>6,79B</td>
</tr>
<tr>
<td>RHI-BMTC</td>
<td>6,59B</td>
<td>756,92M</td>
</tr>
<tr>
<td>SGC-FIT</td>
<td>253,18M</td>
<td>1,20B</td>
</tr>
<tr>
<td>SON-BIS</td>
<td>6,20B</td>
<td>-</td>
</tr>
<tr>
<td>TJX-PDM</td>
<td>61,30B</td>
<td>2,54B</td>
</tr>
<tr>
<td>VLGEA-YUM</td>
<td>380,87M</td>
<td>31,07B</td>
</tr>
<tr>
<td><strong>Bankrupt pairs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHA-BJZ</td>
<td>39,99B</td>
<td>52,96M</td>
</tr>
<tr>
<td>EEI-ATAX</td>
<td>47,43M</td>
<td>427,78M</td>
</tr>
<tr>
<td>USAK-OPOF</td>
<td>98,97M</td>
<td>110,80M</td>
</tr>
<tr>
<td>IHT-OC</td>
<td>14,13M</td>
<td>5,32B</td>
</tr>
<tr>
<td>UXI-ALTR</td>
<td>-</td>
<td>2,73B</td>
</tr>
</tbody>
</table>

*Table 5: Market capitalization for pairs in CRSP 2018-2019.*

<table>
<thead>
<tr>
<th>Pair</th>
<th>Correlation</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOR-KOSS</td>
<td>-0.050</td>
<td>0.404</td>
</tr>
<tr>
<td>ADRE-FLO</td>
<td>-0.071</td>
<td>0.232</td>
</tr>
<tr>
<td>AMSWA-PGC</td>
<td>0.150</td>
<td>0.011</td>
</tr>
<tr>
<td>BOSC-NBN</td>
<td>-0.060</td>
<td>0.319</td>
</tr>
<tr>
<td>EBTC-TKC</td>
<td>-0.242</td>
<td>0.000</td>
</tr>
<tr>
<td>EIS-HBNC</td>
<td>-0.082</td>
<td>0.171</td>
</tr>
<tr>
<td>EXI-SQM</td>
<td>-0.114</td>
<td>0.055</td>
</tr>
<tr>
<td>FDD-ETY</td>
<td>-0.109</td>
<td>0.067</td>
</tr>
<tr>
<td>FISI-MFL</td>
<td>0.019</td>
<td>0.755</td>
</tr>
<tr>
<td>FLXS-INDB</td>
<td>-0.104</td>
<td>0.082</td>
</tr>
<tr>
<td>LBIX-NKG</td>
<td>-0.028</td>
<td>0.736</td>
</tr>
<tr>
<td>MAT-FELE</td>
<td>0.231</td>
<td>0.000</td>
</tr>
<tr>
<td>MCN-CUR</td>
<td>0.087</td>
<td>0.146</td>
</tr>
<tr>
<td>MOG-FXF</td>
<td>-0.010</td>
<td>0.862</td>
</tr>
<tr>
<td>Pair</td>
<td>Correlation</td>
<td>p</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>NUM-APT</td>
<td>0.005</td>
<td>0.933</td>
</tr>
<tr>
<td>PAI-CRESY</td>
<td>0.020</td>
<td>0.743</td>
</tr>
<tr>
<td>PCYO-OVBC</td>
<td>-0.094</td>
<td>0.117</td>
</tr>
<tr>
<td>PDM-FOE</td>
<td>-0.184</td>
<td>0.002</td>
</tr>
<tr>
<td>PGF-KEP</td>
<td>-0.162</td>
<td>0.006</td>
</tr>
<tr>
<td>RHI-BMTC</td>
<td>-0.003</td>
<td>0.957</td>
</tr>
<tr>
<td>SGC-FIT</td>
<td>0.046</td>
<td>0.440</td>
</tr>
<tr>
<td>SON-BIS</td>
<td>-0.040</td>
<td>0.501</td>
</tr>
<tr>
<td>TJX-PDM</td>
<td>-0.047</td>
<td>0.433</td>
</tr>
<tr>
<td>VLGEA-YUM</td>
<td>-0.160</td>
<td>0.007</td>
</tr>
<tr>
<td>UXI-ALTR</td>
<td>-0.111</td>
<td>0.161</td>
</tr>
<tr>
<td>USAK-OPOF</td>
<td>0.035</td>
<td>0.597</td>
</tr>
<tr>
<td>EEI-ATAX</td>
<td>0.084</td>
<td>0.299</td>
</tr>
<tr>
<td>IHT-OC</td>
<td>-0.793</td>
<td>0.000</td>
</tr>
<tr>
<td>CHA-BJZ</td>
<td>-0.219</td>
<td>0.059</td>
</tr>
</tbody>
</table>

Table 6: Correlation of portfolio returns and one-week aggregate volume.
Figure 5 shows the portion of non-bankrupt pairs we had long vs. short over the period.

7.5 Appendix E: Case AMD-Intel

We make a simple long-short strategy on AMD and Intel stocks based on historical data.

$H_0$: There exists no mapping

$$ f(P(AMD)_{t-n}, ..., P(AMD)_{t-1}, P(INTC)_{t-n}, ..., P(INTC)_{t-1}) = g'(AMD)_t, g'(INTC)_t $$

where $g(x)$ is a predictor of

$$ g(x) = \begin{cases} 
1 & \text{if } P(x)_t \geq P(x)_{t-1} \\
0 & \text{otherwise.} 
\end{cases} $$

7.5.1 Data acquisition

**Training Data.** We take tickers ‘INTC’, ‘AMD’, from Quandls WIKI PRICES API from 2016-01-01 to 2017-08-01, normalize to $x_{t=0} = 1$ and take the first differences. For training we use a temporal stochastic gradient descent method in batch generation, sampling a vector with length batchsize of random indices from an uniform distribution.
Figure 6: AMD daily returns and predictions

\((0, T - timesteps)\). We will thus not necessarily go through entire epochs of the training data, rather sampling an array of shape \((\text{time\_steps}, \text{batch\_size}, \text{x\_dim})\) for each batch where \(t_{\text{start}}\) for each example in the batch is a random integer in the range.

**Testing data.** We obtain the testing data from the same source with the same methodology, obtaining 2017-08-02 to 2018-27-03, predicting the last trading 120 days of that period.

### 7.5.2 Results

We can see the model mostly converges to a mean of close to zero after 30 000 batches. This was characteristic over multiple attempts with different hyperparametrization. This is exemplified by our last iteration, where the \(\hat{g}(\text{INTC})\) was correct 65 times and \(\hat{g}(\text{AMD})\) 58 times out of 120. We fail to reject the null hypothesis.
Figure 7: Intel daily returns and predictions