Relatório 1 - Redes neurais do tipo LSTM para detecção de pontos de mudança em séries temporais de alta frequência

ALUISIO AMORIM C.[1] and MENDONCA, J. RICARDO G. *[1]

[1]EACH - USP

Março 2024

Resumo

Este relatório descreve a implementação e avaliação de redes neurais do tipo LSTM para detecção de pontos de mudança em séries temporais de alta frequência, com foco na previsão de preços de ações no mercado financeiro. Inicialmente, foi realizada uma revisão dos conceitos básicos de redes neurais, incluindo neurônios, funções de ativação e otimização, seguida pelo estudo específico de redes neurais recorrentes (RNNs) e redes neurais do tipo LSTM. O projeto abrangeu a implementação de modelos a partir do zero (*from scratch*), seguida pela utilização de bibliotecas conhecidas de aprendizado de máquina. Os dados históricos do índice IBOVESPA foram utilizados para treinar e testar o modelo, com uma abordagem de divisão dos dados que minimizou a correlação entre os conjuntos de treinamento e teste. A técnica de busca em grade foi aplicada para encontrar os hiperparâmetros ideais do modelo, resultando em um modelo com uma acurácia satisfatória na previsão dos preços de fechamento ajustado. Os resultados obtidos indicam um bom desempenho, embora ainda haja espaço para melhorias, especialmente em relação à frequência da série temporal e à capacidade de operação em tempo real. A

*Orientador

próxima etapa do trabalho incluirá o endereçamento dessas questões para tentar aprimorar ainda mais a precisão da previsão e explorar a detecção de pontos de mudança na série temporal.

Todas as etapas do desenvolvimento, resultados e blocos de código podem ser encontrados, em detalhes, no seguinte Jupyter Notebook: https://github.com/Amorim33/lstm-book/blob/main/predicting_stock_prices.ipynb. Recomendamos a consulta a este notebook no GitHub para uma visão clara do progresso do projeto até o momento.

Abstract

This report describes the implementation and evaluation of LSTM neural networks for change point detection in high-frequency time series, focusing on predicting stock prices in the financial market. Initially, a review of the basic concepts of neural networks was conducted, including neurons, activation functions and optimization, followed by the specific study of recurrent neural networks (RNNs) and LSTM neural networks. The project involved the implementation of LSTM models from scratch, followed by the use of well-known machine learning libraries. Historical data from the IBOVESPA index was used to train and test the model, with an approach to data splitting that minimized the correlation between training and test sets. Grid search technique was applied to find the optimal hyperparameters of the model, resulting in a model with satisfactory accuracy in predicting adjusted closing prices. The results indicate good model performance, although there is still room for improvement, especially regarding the frequency of the time series and the ability to operate in real time. The next stage of the work will address these improvements to further enhance prediction accuracy and explore change point detection in the time series.

All development steps, results and code blocks can be found, in detail, in the following Jupyter Notebook: https://github.com/Amorim33/lstm-book/blob/main/predicting_stock_prices.ipynb. We recommend checking out this notebook on GitHub for a clear view of the project's progress to date.

1. Introdução

As redes neurais recorrentes (RNNs) têm sido amplamente utilizadas em problemas de séries temporais devido à sua capacidade de lidar com dependências temporais. No entanto, as

RNNs tradicionais enfrentam dificuldades em capturar informações de longo prazo devido ao problema do gradiente que desaparece ou explode [5]. As redes neurais do tipo *Long-short Term Memory* (LSTM) foram introduzidas para superar essas limitações, permitindo que informações relevantes sejam mantidas por períodos prolongados.

Nas redes neurais do tipo LSTM, cada *unidade*[(a)] possui uma estrutura mais complexa do que as unidades em RNNs tradicionais. Elas incluem três "portões"(gates) - o portão de esquecimento (forget gate), o portão de entrada (input gate) e o portão de saída (output gate) - que controlam o fluxo de informações dentro da unidade.

O *forget gate* decide quais informações devem ser descartadas da unidade de memória. O *input gate* regula a adição de novas informações à unidade de memória, enquanto o *output gate* controla a saída da informação da unidade de memória para a próxima etapa da rede.

Esses mecanismos de portões permitem que as LSTMs aprendam a manter informações relevantes por longos períodos de tempo, enquanto descartam informações menos importantes. Isso resolve o problema do gradiente que desaparece ou explode nas RNNs tradicionais, pois as LSTMs podem manter o gradiente estável ao longo do tempo, permitindo um aprendizado mais eficaz de dependências de longo prazo em séries temporais. Características essas, que sugerem um eventual questionamento: é possível, com essas redes, prever os preços das ações no mercado financeiro?

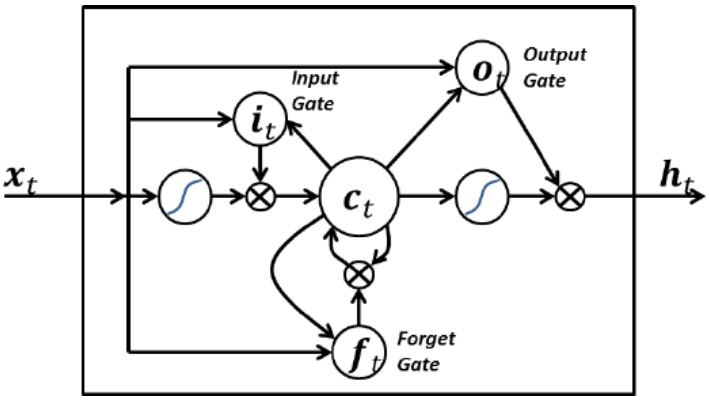

Figura 1: Esquema de uma célula LSTM

Os preços das ações no mercado financeiro são frequentemente modelados como séries temporais de alta frequência devido à sua natureza dinâmica e volátil. Essas séries temporais

[(a)] No contexto de redes neurais, "unidades"geralmente se refere aos neurônios artificiais (perceptrons), que são os blocos de construção básicos dessas redes.

representam a evolução dos preços ao longo do tempo, com observações frequentes e pequenos intervalos entre elas.

No que diz respeito aos mercados de ações, segundo [2], além de sua complexidade e dinamismo inerentes, tem havido um debate constante sobre a previsibilidade dos retornos das ações. A hipótese do Mercado Eficiente, introduzida por [4], define que o preço atual de um ativo sempre reflete instantaneamente todas as informações disponíveis anteriormente para ele. Existe também a hipótese do Passeio Aleatório [1], que afirma que o preço de uma ação muda independentemente de sua história, em outras palavras, o preço de amanhã dependerá apenas das informações de amanhã, independentemente do preço de hoje. Essas duas hipóteses determinam que não há meios de prever com precisão o preço de uma ação. Além disso, [3] realizou uma série de experimentos mostrando que uma estratégia aleatória pode superar alguns dos métodos mais clássicos de negociação técnica, como a Convergência e Divergência de Médias Móveis (MACD) e o Índice de Força Relativa (RSI).

Tendo em vista os severos desafios associados ao tema, o objetivo deste trabalho é averiguar se essas redes neurais do tipo LSTM são capazes de detectar pontos de mudançana nas séries temporais de alta frequência dos preços de ações no mercado financeiro.

2. Materiais e Métodos

Inicialmente, busquei entender os conceitos básicos sobre redes neurais, tipos de aprendizado e modelos que antecedem as redes neurais do tipo LSTM. O primeiro passo foi estudar a história da famosa área de Inteligência Artifical, termo cunhado por John McCarthy em 1956 [6]. A leitura do material de revisão da história anotada da IA moderna e do aprendizado profundo [6], deu-me o contexto necessário dos desdobramentos da área desde a publicação da regra da cadeia por Gottfried Wilhelm Leibniz em 1676, até as atuais redes neurais convolucionais e recorrentes.

Em seguida, aprofundei-me em alguns desses conceitos básicos, como a definição de neurônio, a função de ativação, a função de custo, a função de otimização, *feedforward* e *backpropagation*. Para isso, utilizei séries de videoaulas online e gratuitas, como as do canal StatQuest with Josh Starmer [7], que me ajudaram a entender os conceitos de maneira mais didática e prática.

Razoavelmente confortável com os conceitos básicos, enfim chegou o momento de

debruçar-me sobre as redes neurais recorrentes (RNNs) e as redes neurais do tipo LSTM. Em busca de entender os fundamentos dessas redes, consumi diversas fontes de informação, como artigos, posts de blogs, vídeos e códigos-fonte de bibliotecas conhecidas; a minha ideia inicial era implementar uma rede neural do tipo LSTM do zero, sem o uso de bibliotecas de aprendizado de máquina, como TensorFlow ou PyTorch. Iniciei a implementação no seguinte repositório: https://github.com/Amorim33/lstm-book, utilizando um Jupyter Notebook https://github.com/Amorim33/lstm-book/blob/main/lstm-from-scratch.ipynb, consegui algum progresso: implementei as funções de ativação *sigmoid*, *tanh* e *softmax*, também implementei a função de inicialização dos pesos *Xavier* e por fim, contrui a classe *LSTM*, que implementa a rede neural do tipo LSTM e contém métodos para *forward pass*, *backward pass*, treino, teste e reset dos *gates*. A classe *LSTM* implementa todos os *gates* de uma rede neural do tipo LSTM, como o *forget gate*, *input gate*, *output gate*, *cell state* e o *hidden state*. Entretanto, em um dado momento do desenvolvimento, percebi que a implementação do zero possivelmente não seria a melhor abordagem para o problema proposto, pois demandaria muito tempo e esforço, e poderia não ser eficiente.

Tendo em vista os objetivos do projeto e o conhecimento já obtido, decidi avançar a etapa de implementação do zero e utilizar bibliotecas de redes neurais conhecidas.

Para averiguar se as redes neurais do tipo LSTM são capazes de detectar pontos de mudança nas séries temporais de alta frequência dos preços de ações no mercado financeiro, utilizei os dados históricos do índice IBOVESPA (`^BVSP`), disponíveis no Yahoo Finance https://finance.yahoo.com/quote/%5EBVSP/history/.

Foram considerados os dados diários de abril de 1993 a março de 2024. O IBOVESPA é o principal indicador do desempenho médio das ações listadas na B3 (Bolsa de Valores do Brasil), que é a principal bolsa de valores do país. Portanto, ele reflete as tendências e movimentos do mercado de ações brasileiro como um todo. Por ser um índice composto por ações de empresas líderes e de grande porte, o IBOVESPA tende a ter alta liquidez e volume de negociação. Isso significa que há um grande número de transações ocorrendo diariamente, o que torna os dados mais confiáveis e robustos para análises de séries temporais.

O primeiro passo foi a preparação dos dados, que consistiu em carregar o *dataset* do índice IBOVESPA em um *DataFrame* do Pandas, e realizar uma análise exploratória; essa etapa de preparação foi simples, devido a natureza dos dados, que já estavam em um formato adequado para a análise e não possuíam “buracos”, uma vez que o *dataset* contém todos os

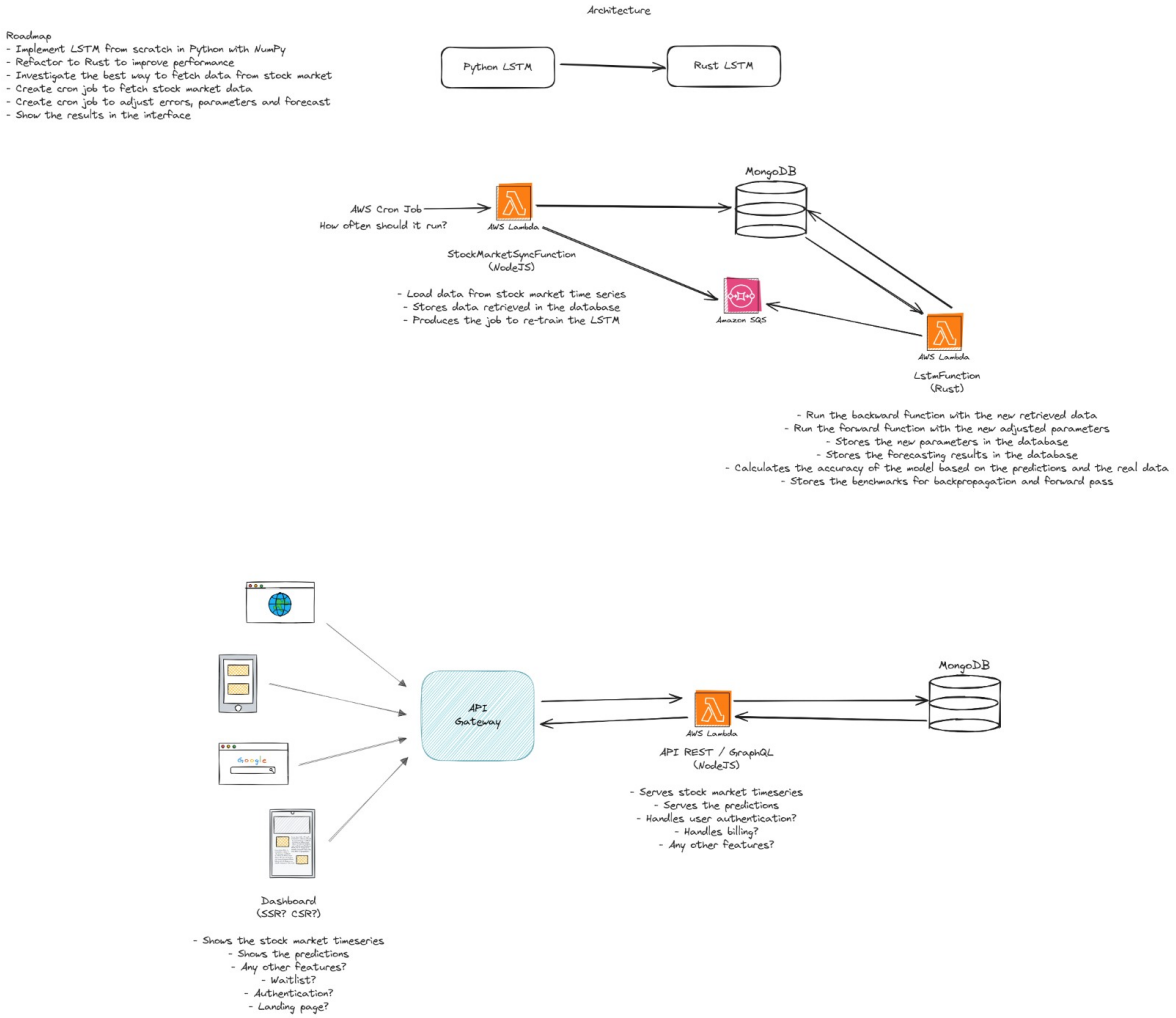

Figura 2: Roadmap do projeto e arquitetura

dias úteis do mercado financeiro brasileiro. Em seguida, a variável alvo foi definida como a coluna *Adj Close*, que representa o preço de fechamento ajustado das ações; esta parece ser a variável mais adequada porque os ajustes que ela carrega refletem o pagamento de dividendos, desdobramentos e agrupamentos de ações, e outras mudanças que afetam o preço de uma ação.

Para a construção do modelo de previsão, foi dividido o conjunto de dados em partições de treino e teste.

A escolha do “ponto de corte”foi um fator crítico para a qualidade do modelo, pois, tendo em vista o objetivo de criar um modelo capaz de “prever o futuro”, é importante que os dados de treino e teste tenham baixa correlação entre si, mas que sejam representativos do conjunto de dados.

Para evitar o problema de overfitting, os dados foram particionados de modo que a

correlação entre os conjuntos de treino e teste fosse mínima. Logo, foi implementada uma função para encontrar o ponto de corte ideal para a divisão dos dados, chamada `get_split_point`. Considerando a seguinte fórmula:

$$C(t) = E[X(s)X(t+s)] - E[X(s)]E[X(t+s)],$$

onde

- $C(t)$ é a autocorrelação de X para um *lag* t;
- t é o *lag* (atraso) entre variáveis aleatórias $X(s)$ e $X(t+s)$;
- s é o instante de tempo;
- E é o operador de esperança matemática (valor esperado);
- $X(t), t = 1,2,\cdots$ é a série temporal.

A autocorrelação é uma medida de correlação entre os valores da série temporal em diferentes instantes de tempo, ou seja, é uma medida de correlação entre a série temporal e ela mesma deslocada no tempo.

Para simplificar a análise, foi assumida a hipótese de que a série temporal é estacionária, ou seja, que a média e a variância não variam com o tempo. Assim, temos que $E[X(s)] = \mu$ e $E[X(t+s)] = \mu$, logo:

$$C(t) = E[X(s)X(t+s)] - E[X(s)]^2$$

A função `get_split_point` retorna o valor do lag que minimiza o valor absoluto da autocorrelação, ou seja, o valor do lag que maximiza a independência entre as séries temporais de treino e teste.

Ao encontrar o primeiro resultado menor ou igual a zero (ou seja, próximo a zero), é retornado o valor do lag atual, pois correlações extremamente negativas podem indicar que os conjuntos de dados estão inversamente correlacionados, o que também não é desejável.

Para melhorar a acurácia do modelo, foram normalizados os dados de treino e teste. A normalização transforma os dados no intervalo entre 0 e 1, sem distorcer as diferenças nas faixas de valores. Ou seja, ela não retira os *outliers* (valores extremos).

Para a modelagem, os dados de treino e teste foram preparados de forma que pudessem ser utilizados para treinar e testar o modelo. O treinamento foi feito da seguinte maneira:

1. Os dados de treino foram divididos em `X_train` e `y_train`, onde `X_train` é uma matriz contendo sequências de `seq_length` valores da série temporal e `y_train` é um vetor contendo os valores subsequentes na série temporal(*labels*). e.g. `X_train[0]` contém os valores de `X[0]` a `X[seq_length-1]` e `y_train[0]` contém o valor de `X[seq_length]`;

2. O mesmo foi feito para os dados de teste, que foram utilizados para avaliar a acurácia do modelo, definindo `X_test` e `y_test`.

Para a criação da rede neural, foi utilizada a classe `Sequential` da biblioteca Keras. Com o `keras.Sequential`, foi possível criar facilmente uma rede neural empilhando camadas, passando uma lista para o construtor. Isso torna a criação de redes neurais simples e intuitiva, especialmente para redes com uma única entrada, uma única saída e uma sequência linear de camadas intermediárias.

A rede neural foi composta por 3 camadas:

1. A primeira camada LSTM contém `units` unidades (ou neurônios) e espera uma entrada tridimensional com a forma `(seq_length, 1)`. Isso significa que ela espera sequências de comprimento `seq_length` com uma única característica em cada passo de tempo. Essa característica no nosso caso é o preço de fechamento ajustado da ação em um dado dia.

2. A segunda camada LSTM também possui `units` unidades.

3. A última camada é uma camada `Dense` com um único neurônio, que é usada para produzir a saída da rede. Como, podemos comparar o objetivo do projeto com uma tarefa de regressão (prever um único valor), uma única saída é o suficiente.

Para encontrar os *hiperparâmetros*[(b)] mais adequados para o modelo, foi utilizada a técnica de `grid search`, que consistiu em testar todas as combinações possíveis de hiperparâmetros (definidas em um objeto `param_grid`) e escolher a que resultou no melhor desempenho do modelo.

O estimador utilizado para a busca em grade foi o `KerasRegressor`, que é uma classe que encapsula o modelo de rede neural criado com o Keras, permitindo que ele seja

[(b)] Os hiperparâmetros de um modelo são configurações que não são aprendidas durante o treinamento do modelo, mas que afetam diretamente o processo de treinamento e o seu desempenho final. Esses hiperparâmetros são definidos antes do início do treinamento

utilizado como um estimador[(c)] do `scikit-learn`[(d)].

Por fim, o modelo foi criado, compilado e treinado utilizando as três camadas descritas anteriormente e os hiperparâmetros retornados pela busca em grade.

```
model = tf.keras.Sequential([
    keras.layers.LSTM(units=256, input_shape=(seq_length, 1),
        return_sequences=True),
    keras.layers.LSTM(256),
    keras.layers.Dense(1)
])
model.compile(optimizer='rmsprop', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=150, batch_size=16, verbose=2)
```

Figura 3: Modelo em Python

3. Resultados

O modelo treinado foi utlizado para "prever" os valores de fechamento ajustado. Para avaliar a sua acurácia, foi utilizada a técnica de *backtest*[(e)]; como o conjunto de dados foi dividido em "treino"e "teste", podemos comparar os valores reais contidos no conjunto de "teste"e os valores preditos pelo modelo.

A estatística `MAPE`, também conhecida como `MAPD` (mean absolute percentage deviation), é uma medida de acurácia de previsão de um modelo.

(c) Em scikit-learn, um estimador é qualquer objeto que pode estimar parâmetros a partir de um conjunto de dados. Isso inclui algoritmos de aprendizado de máquina, como modelos de regressão linear, árvores de decisão e máquinas de vetores de suporte (SVMs), entre outros.

(d) O scikit-learn é uma biblioteca de aprendizado de máquina em Python que oferece uma variedade de algoritmos de aprendizado supervisionado e não supervisionado, ferramentas para pré-processamento de dados, avaliação de modelo e outras funcionalidades relacionadas ao aprendizado de máquina.

(e) Backtest é um método utilizado em finanças e investimentos para avaliar a eficácia de uma estratégia de negociação ou investimento. Consiste em aplicar uma estratégia a dados históricos para simular como teria sido o desempenho dessa estratégia no passado.

```
predicted_values = model.predict(X_test)
```

Figura 4: Método de previsão

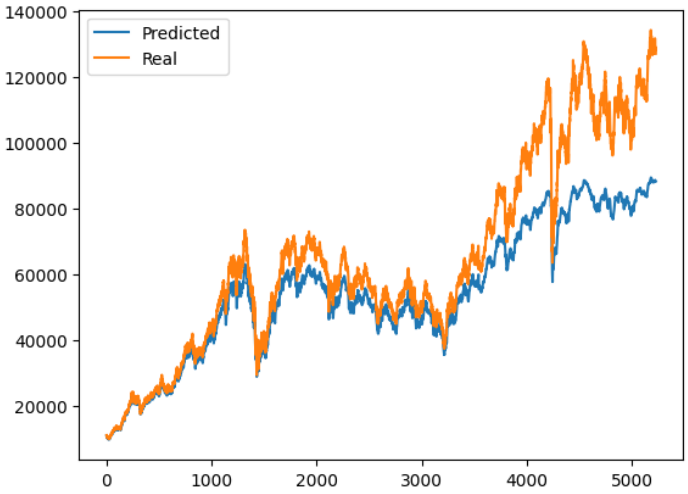

Figura 5: Valores preditos X valores reais

Ela é calculada como a média do valor absoluto das diferenças entre as previsões do modelo e os valores reais, dividido pelo valor real.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

A estatística `MAPE` é comumente utilizada em problemas de regressão, e foi escolhida por ser uma medida de erro relativo e de fácil interpretação.

Os resultados obtidos foram satisfatórios, com um `MAPE` de 11.59%, o que indica que o modelo é bom para prever os preços de fechamento ajustado do índice IBOVESPA.

4. Discussão e Conclusões

Todas as etapas do processo, resultados e blocos de código podem ser encontrados, em detalhes, no seguinte Jupyter Notebook: https://github.com/Amorim33/lstm-book/blob/main/predicting_stock_prices.ipynb.

Os resultados obtidos nesta primeira etapa do projeto são promissores, e indicam que vale a pena investir em abordagens mais sólidas e escaláveis, aproveitando-se de um grande espaço para melhorias.

MAPE	Interpretation
<10	Highly accurate forecasting
10-20	Good forecasting
20-50	Reasonable forecasting
>50	Inaccurate forecasting

Source: Lewis (1982, p. 40)

Figura 6: Interpretação MAPE

- A frequência da séria temporal é baixa, intervalos diários não são suficientes para explicar integralmente as variações no preço de uma ação;
- O modelo de previsão, idealmente, deve operar em tempo real, sendo capaz de prever o preço de fechamento ajustado das ações com base nos dados mais recentes;
- É necessário explorar a predição de pontos de mudança na série temporal, que podem indicar oportunidades de compra e venda de ações.

A segunda etapa do trabalho compreende o enderaçamento desses pontos de melhoria.

Referências

[1] Alessio E Biondo, Alessandro Pluchino, Andrea Rapisarda, and Dirk Helbing. Are random trading strategies more successful than technical ones? *PLoS ONE*, 8:e68344, Jul 2013.

[2] Eugene F Fama and Burton G Malkiel. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):383–417, 1970.

[3] Andrew Lo and Craig MacKinlay. *A Non-Random Walk Down Wall Street*. Princeton University Press, Princeton, NJ [u.a.], 1999.

[4] Burton G Malkiel. *A Random Walk Down Wall Street*. Norton, New York, 1973.

[5] António H. Ribeiro, Koen Tiels, Luis A. Aguirre, and Thomas Schön. Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2370–2380. PMLR, 26–28 Aug 2020.

[6] Jürgen Schmidhuber. Annotated history of modern AI and deep learning. *Available online: https://people.idsia.ch/j̃uergen/deep-learning-history.html*, 2022.

[7] Josh Starmer. Neural networks / deep learning statquest with Josh Starmer. YouTube playlist, ongoing.