Faculdade de Tecnologia de São Paulo – FATEC

Curso de Ciência de Dados

Rede Neural LSTM para Previsão de Preços de Portfólios de Ações: Uma Abordagem de Séries Temporais com Análise PCA

Aluno: Murilo Reis de Jesus

Professor: Dr. Alexandre Garcia de Oliveira

São Paulo
Junho de 2025

Resumo

Este trabalho apresenta uma análise técnico-acadêmica de um modelo de Rede Neural Recorrente: Long Short-Term Memory (LSTM) aplicado à previsão de preços de um portfólio composto por ativos como AAPL, MSFT e GOOGL. Utilizando dados históricos semanais de três anos, a abordagem envolve normalização via z-score, janelas deslizantes para modelagem sequencial e treinamento com função de perda Huber. Complementarmente, são realizadas análises com PCA para interpretação das variáveis latentes e evidência de suas uniformidades. O modelo demonstrou capacidade de prever com boa aderência ao comportamento real do mercado, sugerindo potencial para tomada de decisões automatizadas.

1 Introdução

A previsão de preços de ativos financeiros é uma tarefa desafiadora em aprendizado de máquina devido à sua natureza estocástica, volátil e altamente não linear. Modelos tradicionais, como regressões lineares e ARIMA (Autoregressive Integrated Moving Average), enfrentam dificuldades em capturar padrões de longo prazo. LSTMs, por outro lado, são capazes de modelar dependências temporais de forma mais eficaz, já que conseguem generalizar mais facilmente séries temporais com muitos outliers.

Nesse contexto, modelos baseados em redes neurais recorrentes, especialmente as LSTMs (Long Short-Term Memory), têm ganhado destaque por sua capacidade de aprender representações temporais complexas e de longo alcance em séries temporais. As LSTMs se diferenciam das redes recorrentes tradicionais (RNNs) por incorporarem mecanismos internos de memória e controle (portas de entrada, esquecimento e saída), permitindo que o modelo retenha ou descarte informações ao longo do tempo com mais eficiência. Isso as torna especialmente adequadas para contextos em que padrões relevantes podem estar distribuídos ao longo de longas sequências temporais, como ocorre frequentemente no comportamento de ativos financeiros.

2 Descrição Técnica do Modelo

O LSTM (Long Short-Term Memory) é um tipo especializado de Rede Neural Recorrente ou *Recurrent Neural Network* (RNN), projetado para trabalhar com dados sequenciais, como séries temporais. As RNNs diferem das redes neurais tradicionais por possuírem conexões recorrentes que permitem o processamento de informações em sequência, ou seja, o modelo leva em conta não apenas a entrada atual, mas também o estado anterior, possibilitando a modelagem de dependências temporais. No entanto, RNNs convencionais enfrentam dificuldades em aprender dependências de longo prazo devido a problemas como o desaparecimento ou explosão do gradiente durante o treinamento.

Para superar essas limitações, foi desenvolvido o LSTM, que incorpora um mecanismo interno baseado em "portas" (entrada, esquecimento e saída) que regulam o fluxo de informação ao longo do tempo. Isso permite que o modelo retenha informações relevantes por períodos longos e descarte o que for irrelevante. Seu nome, Long Short-Term Memory, reflete exatamente essa capacidade de manter na memória tanto informações de curto quanto de longo prazo.

O formato de entrada aceito por um LSTM é tridimensional, com a estrutura *(samples, timesteps, features)*. Cada amostra (*sample*) é uma sequência temporal, os *timesteps* representam o número de períodos usados como contexto, e *features* corresponde ao número

de variáveis por período (no caso deste trabalho, o preço de fechamento). Essa estrutura é necessária para que o modelo reconheça padrões e relações ao longo do tempo.

Internamente, o LSTM realiza cálculos vetoriais em cada passo temporal t, manipulando dois estados principais: o vetor de estado da célula C_t e o vetor de estado oculto h_t. Esses cálculos são definidos pelas seguintes equações:

- **Porta de esquecimento:** define o que será descartado da célula de memória.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- **Porta de entrada:** controla quais novas informações serão armazenadas.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- **Atualização do estado da célula:** combina as informações antigas e novas.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

- **Porta de saída:** define o que será emitido como saída naquele instante.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \odot \tanh(C_t)$$

Nessas equações:

- x_t é a entrada atual (por exemplo, o preço de fechamento da semana);
- h_{t-1} é o estado oculto anterior;
- $\sigma =$

$$\frac{1}{1 + e^{-x}}$$

  é a função sigmoide, que restringe os valores entre 0 e 1;

- $\odot =$

$$b = [a_1 b_1, a_2 b_2, \ldots, a_n b_n]$$

  representa o produto elemento a elemento (Hadamard);

- W e b são os pesos e vieses aprendidos pelo modelo durante o treinamento.

Durante o processo de treinamento, os valores de W e b são ajustados iterativamente com base no erro da previsão, no caso desse modelo, é ajustado com base em Hubert Loss.

De forma simplificada, a célula de memória (Ct) armazena informações ao longo do tempo e é passada para a próxima célula. A porta de esquecimento (ft) decide quanto da memória anterior será mantido, enquanto a porta de entrada (it) adiciona novas informações. A porta de saída (ot) controla a quantidade de memória que será utilizada na saída atual.

2.1 Coleta e Tratamento/Preparação dos Dados

A base de dados foi construída com informações históricas de preços semanais de fechamento das ações, valores finais pelos quais uma ação foi negociada ao término do pregão de um determinado período temporal. As ações consideradas foram AAPL (Apple), MSFT (Microsoft) e GOOGL (Google).Para isso, foi utilizada a biblioteca `yfinance`, que oferece acesso programático a dados históricos da plataforma Yahoo Finance.

O trecho a seguir ilustra a extração dos dados, onde um laço de repetição é empregado para criar um dataset contendo as informações semanais do preço de fechamento das ações ao longo de um período de três anos:

```
dfs = []
for ticker in TICKERS:
    data = yf.Ticker(ticker).history(period="3y", interval="1wk")["Close"]
    data.name = ticker
    dfs.append(data)
```

Em seguida, as séries são unificadas em um único DataFrame com múltiplas colunas:

```
df = pd.concat(dfs, axis=1)
df.dropna(inplace=True)
df["Portfolio"] = df.mean(axis=1)
```

A função `concat` junta todas as colunas por data. Linhas com valores ausentes são eliminadas com `dropna()`. Após isso, cria-se uma nova coluna Portfolio, que representa o valor médio do portfólio com base nos ativos analisados em cada semana. Este é o alvo principal da modelagem.

O vetor de preços do portfólio é extraído e normalizado por meio do z-score, técnica que ajusta os dados para uma distribuição com média zero e desvio padrão um:

```
portfolio_values = df["Portfolio"].values
mean = np.mean(portfolio_values)
std = np.std(portfolio_values)
portfolio_scaled = (portfolio_values - mean) / std
```

Fórmula da Normalização z-score:

$$z = \frac{x - \mu}{\sigma}$$

em que x é o valor de entrada, μ a média e σ o desvio padrão.

Em seguida, são construidas as janelas deslizantes de entrada `X` e os valores alvo `y`. `X` contém sequências de 12 semanas consecutivas (tamanho da janela), e `y` contém o valor imediatamente seguinte a cada sequência. Estre tratamento é necessário, já que o modelo analisa uma série temporal.

```
X, y = [], []
for i in range(len(portfolio_scaled) - WINDOW_SIZE):
    X.append(portfolio_scaled[i:i + WINDOW_SIZE])
    y.append(portfolio_scaled[i + WINDOW_SIZE])
X = np.array(X)
y = np.array(y)
```

Em seguida, os dados são redimensionados para a forma esperada pela camada LSTM, ou seja, como o formato de entrada aceito por uma LSTM segue uma estrutura tridimensional com o seguinte arranjo: *(samples, timesteps, features)*.

```
X = X.reshape((X.shape[0], X.shape[1], 1))
```

Por fim, os dados são divididos em conjuntos de treinamento e teste. A divisão 70/30 é usada para garantir que o modelo seja avaliado em dados não vistos. A estrutura final do conjunto de treinamento é tipicamente (102, 12, 1) e do teste (44, 12, 1) para este dataset, indicando 102 sequências de treinamento e 44 para teste, cada uma com 12 pontos de entrada e 1 variável de input por passo temporal.

No final de todo o processo, temos então o portfólio das médias normalizadas das ações, que seria os preços de fechamento combinados e transformados em uma série temporal que representa a média aritmética normalizada a cada semana. Além de sua redimensão para satisfazer o input esperado pelo modelo LSTM.

```
split = int(len(X) * 0.7)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

2.2 Hiperparâmetros e Mecanismos de Aprendizado

O hiperparâmetro `WINDOW_SIZE` define o número de passos temporais considerados como entrada. No presente estudo, `WINDOW_SIZE = 12`, indica que a rede observará 12 semanas consecutivas para prever o valor da 13ª semana. Esse valor é empiricamente adequado para capturar padrões sazonais de curto prazo, como ciclos trimestrais típicos de finanças corporativas.

Durante o treinamento, a rede ajusta seus parâmetros internos, os pesos, com base nos erros entre a previsão e o valor real. Cada conexão entre neurônios possui um peso que indica a importância daquela entrada. Inicialmente aleatórios, esses pesos são atualizados a cada iteração com base no algoritmo de retropropagação.

A *taxa de aprendizado* (*learning rate*), denotada geralmente por η, é um hiperparâmetro fundamental no processo de otimização do modelo. Ela determina o tamanho dos passos dados durante a atualização dos pesos da rede a cada iteração. Taxas muito altas podem fazer o modelo divergir, pulando os mínimos da função de perda e impedindo a convergência. Por outro lado, taxas muito baixas tornam o aprendizado excessivamente lento e podem levar o modelo a ficar preso em mínimos locais ou estacionar antes de alcançar uma solução ótima. A escolha adequada da taxa de aprendizado é essencial para garantir que o treinamento seja eficiente e estável.

O algoritmo de retropropagação (*backpropagation*) consiste em propagar o erro calculado na saída da rede de volta através das camadas, utilizando o método do gradiente descendente para atualizar os pesos. Isso é feito por meio do cálculo das derivadas parciais da função de perda em relação a cada peso da rede, ou seja, os gradientes. A atualização dos pesos w ocorre de acordo com a seguinte regra:

$$w_{\text{novo}} = w_{\text{antigo}} - \eta \cdot \frac{\partial \mathcal{L}}{\partial w}$$

em que:

- η representa a *taxa de aprendizado* (*learning rate*);

- $\frac{\partial \mathcal{L}}{\partial w}$ é o gradiente da função de perda $\mathcal{L}$ em relação ao peso w acumulado ao longo das etapas de tempo.

Outro hiperparâmetro essencial é o número de épocas (EPOCHS), que indica quantas vezes o conjunto de dados de treino será completamente percorrido. Em geral, mais épocas permitem à rede aprender melhor os padrões dos dados, até um ponto em que há risco de sobreajuste (overfitting). Neste trabalho, o modelo foi treinado por 300 épocas, com controle de validação, o que demonstrou ser suficiente para convergência sem perda de generalização.

2.3 Arquitetura do Modelo LSTM

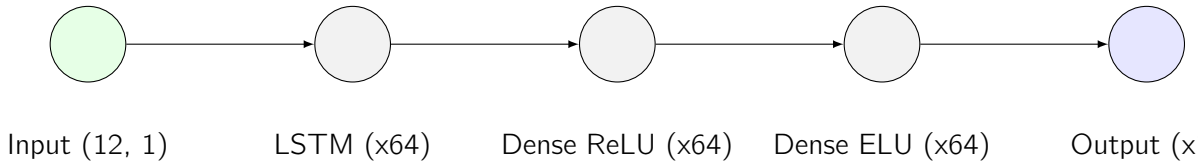

Figure 1: Representação Visual simplificada da arquitetura da rede LSTM com camadas densas subsequentes. Cada nó representa uma transformação vetorial.

Trecho do código onde é declarada a arquitetura do modelo de Rede Neural Recorrente LSTM:

```
model = tf.keras.Sequential([
        tf.keras.layers.LSTM(64, activation='tanh', input_shape=(WINDOW_SIZE
            , 1)),
        tf.keras.layers.Dropout(0.1),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dropout(0.1),
        tf.keras.layers.Dense(64, activation='elu'),
        tf.keras.layers.Dropout(0.1),
        tf.keras.layers.Dense(1)
    ])
```

Cada unidade atribuída as camadas funciona como um neurônio especializado que mantém uma célula de memória e realiza operações de controle via portas (entrada, esquecimento e saída). Essas unidades operam paralelamente, processando a mesma sequência de entrada e extraindo diferentes padrões temporais, como tendências de longo prazo, ciclos sazonais ou flutuações de curto prazo. Por exemplo, a saída da camada LSTM, portanto, é um vetor de 64 dimensões, representando as ativações finais de cada neurônio no tempo mais recente da sequência.

No contexto do modelo proposto, as camadas Dense são utilizadas após a camada LSTM para transformar as representações temporais aprendidas em um espaço mais adequado à tarefa final de regressão, que é a previsão do preço de fechamento. Essa transformação é essencial porque, enquanto a camada LSTM captura as dependências temporais e sequenciais dos dados, as camadas Dense agregam e refinam essas informações, extraindo padrões não lineares adicionais que podem ser decisivos para a precisão da previsão.

2.3.1 Camada Oculta LSTM Função de ativação `Tanh`

A arquitetura da rede conta com uma camada LSTM com 64 unidades (ou neurônios) e ativação "Tanh". A função tangente hiperbólica é definida como:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Ela mapeia os valores de entrada para o intervalo $(-1, 1)$, centrado em zero. Isso contribui para uma convergência mais eficiente do modelo durante o treinamento, ao reduzir o viés de ativações acumuladas e ajuda a capturar variações suaves e simétricas nos dados temporais, como é o caso das séries de preços financeiros.

2.3.2 Camada Oculta Função de ativação `ReLU`

A primeira camada "Dense" é uma camada totalmente conectada com 64 neurônios e ativação ReLU (*Rectified Linear Unit*), que permite o aprendizado de padrões não lineares nas representações temporais extraídas pela LSTM. A função ReLU é definida por:

$$\text{ReLU}(x) = \max(0, x)$$

Ela anula os valores negativos e mantém os positivos, sendo amplamente utilizada em redes profundas por sua simplicidade e eficiência.

2.3.3 Camada Oculta Função de ativação `ELU`

A segunda camada "Dense" é uma camada totalmente conectada com 64 neurônios e ativação ELU (*Exponential Linear Unit*), que permite capturar não linearidades mais complexas nos dados, mantendo uma resposta mais suave e contínua, pois evita o problema de "neurônios mortos" e fornece gradientes pequenos, porém não nulos, para entradas negativas. A função ELU é definida por:

$$\text{ELU}(x) = \begin{cases} x, & \text{se } x > 0 \\ \alpha(e^x - 1), & \text{se } x \leq 0 \end{cases}$$

onde α é um hiperparâmetro positivo, usualmente fixado como $\alpha = 1$.

Diferente da ReLU, a ELU permite a propagação de valores negativos suavizados, o que ajuda a manter a média das ativações próxima de zero. Isso favorece um aprendizado mais estável e robusto. O método ELU, vindo depois, pode refinar os padrões, corrigindo a perda de informação causada pelos zeros da ReLU em entradas negativas.

2.3.4 Camada Dropout

Essa camada serve como regularizador. Ela "desliga" aleatoriamente 10% dos neurônios durante o treinamento, o que reduz o risco de overfitting e melhora a capacidade de generalização da rede:

```
tf.keras.layers.Dropout(0.1)
```

2.3.5 Camada Final

Esta é a camada de saída, com um único neurônio linear, cuja função é prever o valor escalar do portfólio para a próxima semana, definido como:

```
tf.keras.layers.Dense(1)
```

2.3.6 Justificativa da combinação de ativações

A sequência de ativações adotada no modelo foi cuidadosamente escolhida para aproveitar os pontos fortes de cada função:

- `Tanh`: é ideal para modelar dinâmicas temporais em dados sequenciais, como séries históricas de preços;
- `ReLU`: é eficiente para extrair padrões fortes a partir das representações da LSTM;
- `ELU`: melhora a estabilidade e a generalização do modelo ao manter contribuições negativas suaves.

2.4 Descrição do Compile do Modelo LSTM

A etapa de compilação do modelo define os componentes essenciais para o processo de treinamento supervisionado da rede neural. Neste trabalho, foi utilizado o otimizador `Adam`, devido à sua capacidade adaptativa de ajuste dos pesos:

```
model.compile(optimizer='adam', loss=tf.keras.losses.Huber(0.8), metrics=['
    mse','mae', 'mape'])
```

A atualização dos pesos depende da taxa de aprendizado (learning rate), que controla o tamanho do passo dado na direção do gradiente do erro. O otimizador Adam, adapta a taxa de aprendizado automaticamente para cada peso, com base nos momentos acumulados dos gradientes, sendo o seu padrão: 1e-3 ou 0,001.

Como função de perda, foi empregada a `função de Huber`, parametrizada com $\delta = 0{,}8$. Essa função combina as características do erro quadrático médio (MSE) para erros pequenos, garantindo suavidade e sensibilidade aos detalhes, com o erro absoluto médio (MAE) para erros maiores, promovendo maior robustez frente a outliers. O valor de δ define o ponto de transição entre esses dois comportamentos e foi escolhido empiricamente para equilibrar precisão e resistência a valores extremos. Sua função é definida como:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{se } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{caso contrário} \end{cases}$$

Onde:

- $a = y - \hat{y}$ é a diferença entre o valor real e o previsto,
- δ é o ponto que define a transição entre o regime quadrático e o linear.

Durante o treinamento, o modelo monitora três métricas: `erro quadrático médio (MSE)`, `erro absoluto médio (MAE)` e `erro percentual absoluto médio (MAPE)`. O MSE é sensível a grandes erros e útil para penalizar desvios significativos, enquanto o MAE fornece uma medida mais estável e menos afetada por valores atípicos. Já o MAPE oferece uma perspectiva percentual do erro em relação aos valores reais, sendo especialmente relevante para a análise da precisão relativa das previsões em cenários financeiros.

2.4.1 Exemplo das Métricas de Loss

Considere um exemplo em que o valor real de uma observação é $y = 100$ e a previsão realizada pelo modelo é $\hat{y} = 98$:

- **Erro Absoluto Médio (MAE)**:

  O MAE é definido como a média do valor absoluto das diferenças entre os valores reais e previstos:

  $$\text{MAE} = |y - \hat{y}| = |100 - 98| = 2$$

- **Erro Quadrático Médio (MSE)**:

  O MSE é definido como a média do quadrado das diferenças entre os valores reais e previstos:

  $$\text{MSE} = (y - \hat{y})^2 = (100 - 98)^2 = 4$$

- **Erro Percentual Absoluto Médio (MAPE)**:

  O MAPE é definido calcula a média dos erros absolutos entre os valores reais e os valores previstos, expressos como uma porcentagem do valor real.

  $$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| = \left| \frac{100 - 98}{100} \right| \times 100 = 2\%$$

- **Huber Loss**:

  Para este exemplo, temos:

  $$y = 100, \quad \hat{y} = 98, \quad a = 100 - 98 = 2$$

  Como $|a| = 2 > \delta = 0{,}8$, utilizamos a segunda parte da definição da função:

  $$L_\delta(a) = \delta(|a| - \frac{1}{2}\delta) = 0{,}8 \cdot (2 - 0{,}4) = 0{,}8 \cdot 1{,}6 = 1{,}28$$

Portanto, a perda de Huber para este par de valores é igual a **1,28** , demonstrando que, nesse caso, o erro é tratado de forma linear, o que reduz o impacto de outliers em comparação ao MSE.

3 Análise dos Resultados do modelo

Nesta seção, serão apresentados e discutidos os resultados obtidos a partir do modelo LSTM treinado para previsão dos preços médios de um portfólio composto pelas ações *AAPL*, *MSFT* e *GOOGL*. A análise contempla a comparação entre os valores reais e os valores previstos pela rede, utilizando uma janela deslizante.

Além da visualização gráfica da série temporal, também será avaliado o desempenho quantitativo do modelo por meio das métricas de erro, incluindo o erro quadrático médio, o erro absoluto médio e o erro percentual absoluto médio. Essas métricas permitem não apenas quantificar a acurácia do modelo, mas também observar como diferentes tipos de erro são tratados durante o processo de aprendizado, oferecendo uma visão mais completa sobre a robustez e a generalização do modelo em relação aos dados utilizados.

Nestre treinamento, o modelo alcançou as seguintes Métricas:

- **Erro Absoluto Médio (MAE)**: 0.145
- **Erro Quadrático Médio (MSE)**: 0.0324
- **Erro Percentual Absoluto Médio (MAPE)**: 21.933

3.1 Análise do Gráfico de Previsões

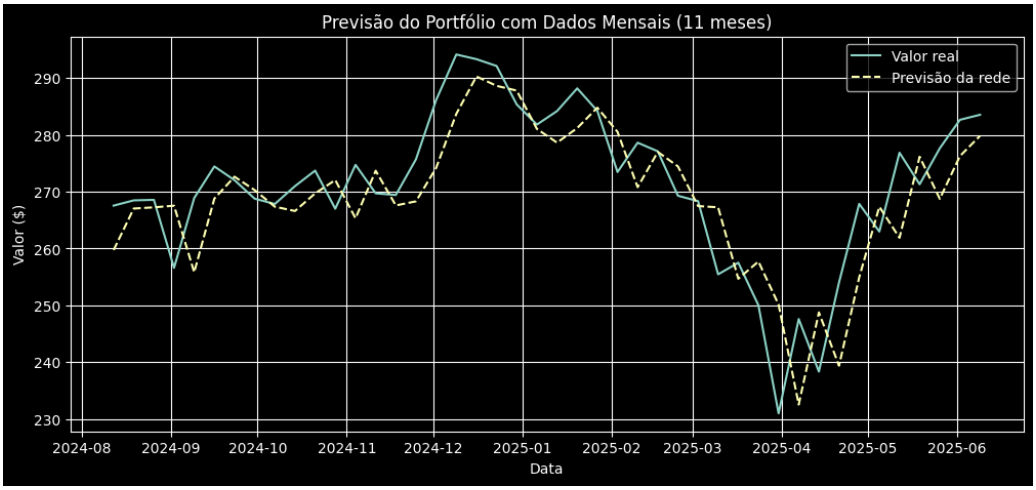

Figure 2: Previsão do Portfólio com Onze Meses (11 meses de previsão)

Como se observa na Figura abaixo, o modelo baseado em LSTM consegue capturar de forma satisfatória a tendência geral do movimento médio das ações. A linha contínua representa os valores reais da média semanal das ações, enquanto a linha tracejada indica as previsões obtidas pelo modelo LSTM sobre os dados de teste. Embora existam variações pontuais entre os valores previstos e os reais, especialmente em momentos de maior oscilação, a sobreposição das curvas indica que a rede neural obteve uma boa capacidade de generalização.

Outro aspecto relevante é que o modelo parece reagir de forma gradual a mudanças bruscas no mercado, suavizando picos acentuados. Isso pode ser atribuído ao uso da média entre os ativos, ao método de Loss usado (Huber), ou então, à natureza da arquitetura LSTM, que preserva informações passadas com mais equilíbrio em relação a flutuações recentes.

3.2 Previsão Estendida do Portfólio

A Figura abaixo considera os dados reais dos últimos 11 meses e uma projeção adicional de quatro semanas. As linhas seguem as mesma descrição da Figura 2, com a única diferença sendoa linha pontilhada vermelha, a qual corresponde à extensão futura feita pelo modelo, que utiliza a última janela de observações para prever iterativamente as semanas subsequentes.

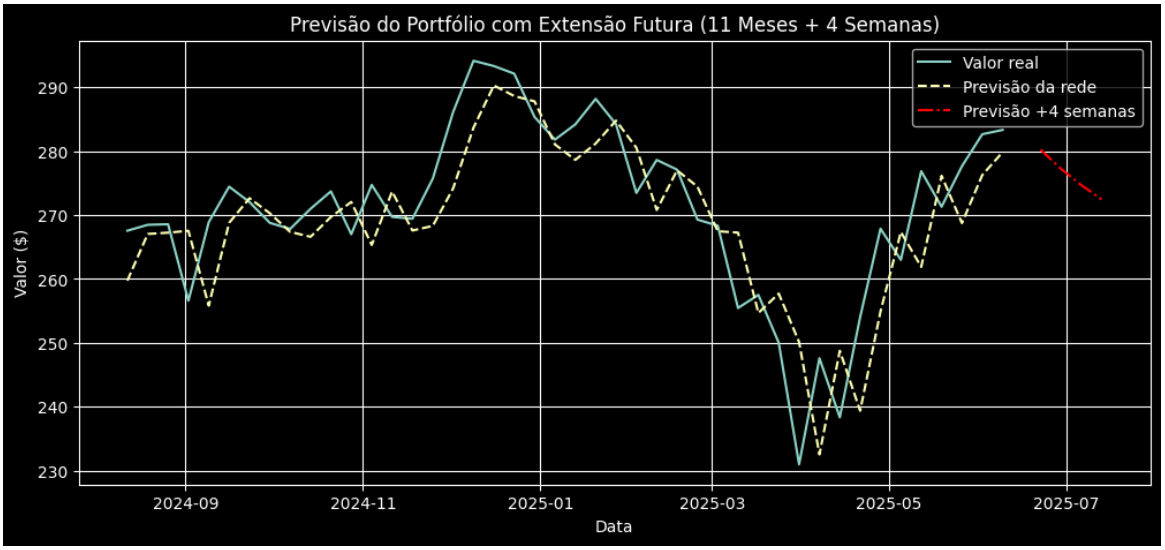

Figure 3: Previsão do Portfólio com Extensão Futura (11 Meses + 4 Semanas)

Observa-se que a rede neural LSTM conseguiu acompanhar com razoável precisão a trajetória da média do portfólio durante o período avaliado. As previsões mantêm-se próximas aos valores reais, inclusive durante períodos de maior volatilidade, como evidenciado entre fevereiro e maio de 2025. Além disso, a projeção adicional para as quatro semanas seguintes revela uma tendência de estabilização, sugerindo que o modelo identificou um possível esgotamento da recuperação observada anteriormente.

A extensão da previsão foi realizada por meio de um processo autorregressivo, em que as previsões anteriores alimentam o modelo para gerar os próximos valores, presente neste trecho:

```
last_window = portfolio_scaled[-WINDOW_SIZE:].tolist()
future_predictions_scaled = []
for _ in range(FUTURE_STEPS):
    input_array = np.array(last_window[-WINDOW_SIZE:]).reshape(1,
       WINDOW_SIZE,1)
    next_pred = model.predict(input_array)[0, 0]
    future_predictions_scaled.append(next_pred)
    last_window.append(next_pred)

future_predictions_real = [p * std + mean for p in future_predictions_scaled
   ]
predictions_str = ', '.join([str(p) for p in future_predictions_real])
```

Embora este método possa propagar erros cumulativos ao longo do tempo, os valores projetados não apresentam desvios abruptos em relação à tendência estabelecida, o que reforça a robustez do modelo ao lidar com séries temporais multivariadas de ativos financeiros.

Neste caso, os valores previstos foram: \$280,66, \$277,59, \$274,99 e \$272,70.

3.3 Análise da Curva de Loss

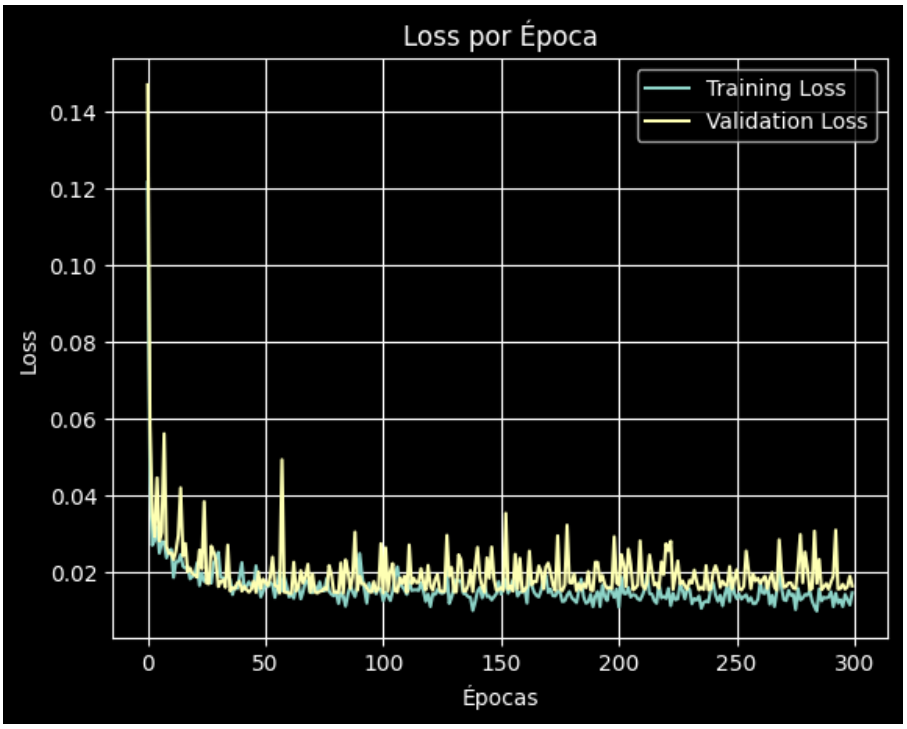

Figure 4: Evolução do Huber Loss durante o treinamento (300 épocas).

O gráfico da Figura 4 apresenta a evolução da função de perda (Huber Loss) ao longo de 300 épocas, tanto para os dados de treinamento quanto para os dados de validação.

Observa-se que, nas primeiras épocas, há uma queda acentuada no valor da loss, indicando que o modelo rapidamente aprende padrões relevantes nos dados. Após aproximadamente 50 épocas, a curva se estabiliza, mantendo valores baixos e relativamente constantes

A curva de *training loss* (linha azul) mostra uma trajetória suavemente decrescente, com baixa variação, o que indica uma boa capacidade de aprendizado do modelo sem sinais evidentes de overfitting. Já a curva de *validation loss* (linha amarela) apresenta oscilações mais intensas ao longo das épocas, o que é esperado em séries temporais devido à variabilidade natural dos dados, mas ainda assim mantém-se próxima à curva de treinamento, demonstrando uma boa generalização.

4 Análise do PCA

Para compreender quais ações exercem maior influência sobre a variação do portfólio composto por três empresas, foi aplicada a técnica de Análise de Componentes Principais (PCA) aos dados históricos dos preços das ações, normalizados por meio do método de padronização StandardScaler, o qual aplica o fit para calcular a média e o desvio padrão de cada variável, e o transform, normaliza os dados, subtraindo a média e dividindo pelo desvio padrão. Isso é necessário para o PCA, pois ele é sensível à escala das variáveis.

Na etapa de PCA, apenas o fit é aplicado, já que a ideia é apenas analisar os componentes principais, não é necessário reduzir a dimensionalidade do conjunto.

O objetivo principal dessa análise foi identificar as variáveis (ações de determinada empresa) que mais contribuem para a variabilidade total observada no portfólio, especialmente no contexto

do primeiro componente principal (PC1), que representa a direção de maior variância nos dados multivariados.

O resultado da decomposição PCA revelou que os coeficientes (ou cargas) do primeiro componente principal foram os seguintes:

```
Contribuição de cada ação no 1º componente principal (PC1):
GOOGL    0.586090
MSFT     0.576954
AAPL     0.568878
```

Figure 5: Contribuição de cada ação no 1º componente principal (PC1)

Esses valores indicam que as três ações possuem contribuições bastante próximas entre si na composição do PC1, todas com coeficientes positivos e de magnitude semelhante. Tal configuração sugere que essas ações tendem a se mover de forma conjunta, isto é, possuem um comportamento altamente correlacionado ao longo do tempo. Do ponto de vista do PCA, isso significa que o primeiro componente captura uma tendência comum de variação entre os ativos, e que nenhuma ação individual domina isoladamente a variância explicada.

Consequentemente, pode-se concluir que todas as ações analisadas contribuem de forma equilibrada para as oscilações gerais do portfólio. Isso está em sincronia com a hipótese inicial de investigar qual ação mais influencia o valor médio do portfólio, indicando, porém, que não há uma única ação dominante: a variação observada é explicada de maneira conjunta pelas três empresas.

5 Conclusão

Conclui-se que o modelo de rede neural recorrente LSTM (Long Short Term Memory) apresenta capacidade consistente para aprender padrões históricos dos preços das ações e projetar cenários futuros de curto prazo com coerência e estabilidade. A estabilidade e convergência das curvas de perda (Loss) indicam que o modelo está bem ajustado aos dados, e a escolha da função de perda de Huber se mostrou adequada por oferecer maior robustez diante de outliers, em comparação ao erro quadrático médio (MSE).

A análise do portfólio por meio da média aritmética normalizada dos preços semanais das ações revelou-se eficiente para aplicação do PCA. Os resultados indicam que as três empresas analisadas contribuem de forma significativa e linear para a variação do portfólio. Dessa forma, ao monitorar o valor médio do portfólio, é possível avaliar com maior clareza as vantagens de investimento em cada uma das ações simultaneamente, facilitando a tomada de decisão estratégica.

Referências

- AIML Community. *O que é ReLU morta (dying ReLU) e por que isso é um problema no treinamento de redes neurais?* Disponível em: https://aiml-com.translate.goog/what-is-dying-relu-or-dead-relu-and-why-is-this-a-problem-in-neural-network-training/?_x_tr_sl=en.

- JOLLIFFE, I. T.; CADIMA, J. *Principal component analysis: a review and recent developments*. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2016. Disponível em: https://doi.org/10.1098/rsta.2015.0202

- LOURENÇO, Alan. *LSTM – Long Short-Term Memory*. Didática Tech, 2021. Disponível em: https://didatica.tech/lstm-long-short-term-memory/.
- MATPLOTLIB. Matplotlib documentation. Disponível em: https://matplotlib.org/stable/plot_types
- SCIKIT-LEARN. Scikit-learn documentation. Disponível em: https://scikit-learn.org
- TENSORFLOW. TensorFlow documentation. Disponível em: https://www.tensorflow.org/api_docs/
- TENSORFLOW *Long Short-Term Memory layer*, Hochreiter 1997. TensorFlow documentation: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM