UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE ECONOMIA
MONOGRAFIA DE BACHARELADO

REDES NEURAIS LSTM E MODELO GARCH:

UMA ABORDAGEM CONJUNTA PARA PREVISÃO DE RETORNOS

GABRIEL CONTARINI
matrícula nº 114053618

ORIENTADORA: Prof.ª Susan Schommer

AGOSTO 2020

As opiniões expressas neste trabalho são da exclusiva responsabilidade do autor

RESUMO

A ideia principal do trabalho é analisar a performance de previsão do retorno de ações, particularmente da ação PETR3. Primeiramente, para prever os retornos futuros, foi usado um tipo de rede neural chamada de redes neurais recorrentes, especificamente neurônios do tipo LSTM. Depois são estimados modelos GARCHs para a série e são usados suas variâncias condicionais como novos *inputs* para as redes neurais, transformando assim um problema de regressão univariado em um problema multivariado usando os mesmos dados iniciais. As redes neurais LSTM, em geral, apresentaram melhores resultados em comparação ao próprio GARCH e as redes neurais com GARCH como *input*.

Palavras-chave: Redes neurais recorrentes. LSTM. GARCH.

ABSTRACT

The main idea from this work is to analyze the forecast performance of a stock return, specially from PETR3. Firstly, it uses the returns from an asset to predict its future returns, using a type of neural network called recurrent neural net, specifically LSTM neurons. After that, it fits some GARCH models to the data and uses its conditional variance as new inputs to the neural nets, transforming a single variable regression problem into a multivariable problem using the same initial data. The LSTM neural nets, in general, yield better forecast performances than the GARCH model and the LSTMs with GARCH as inputs.

Keywords: Recurrent neural networks. LSTM. GARCH.

SUMÁRIO

LISTA DE GRÁFICOS, FIGURAS E TABELAS

INTRODUÇÃO

O campo das redes neurais, apesar de amplamente estudado hoje em dia, ainda é uma área altamente empírica. Diferente da econometria, que tem ampla literatura e se baseia em sua teoria para derivar seus resultados práticos, nas redes neurais quase sempre são obtidos resultados práticos e depois é buscado os porquês teóricos desses resultados, com muitas lacunas ainda a serem preenchidas. A falta de recomendações de boas práticas ao se trabalhar com redes neurais deixa ao pesquisador uma ampla escolha de possibilidades, escolhas que podem levar a bons resultados ou não.

Incumbido desse espírito prático, o trabalho a seguir explora a possibilidade de se unir as duas abordagens, da econometria e das redes neurais. A ideia principal do trabalho é analisar a performance de previsão do retorno de ações, particularmente da ação PETR3. Primeiramente, para prever os retornos futuros, foi usado um tipo de rede neural chamada de redes neurais recorrentes, especificamente neurônios do tipo LSTM. Depois são estimados modelos GARCHs para a série e são usados suas variâncias condicionais como novos *inputs* para as redes neurais, transformando assim um problema de regressão univariado em um problema multivariado usando os mesmos dados iniciais. As redes neurais LSTM, em geral, apresentaram melhores resultados em comparação ao próprio GARCH e as redes neurais com GARCH como *input*.

Abordagens que comparam o poder preditivo de redes neurais com modelos de séries temporais não são incomuns (PORTUGAL, 1995). Porém, este trabalho busca uma outra aproximação mais construtiva de unir as duas abordagens e aplicar ao caso do mercado de ações brasileiro, uma abordagem que também já foi tentada (YIM, 2002). Com o diferencial de usar células LSTM ao invés de neurônios simples.

O trabalho foi dividido em duas grandes seções. A primeira seção faz a exposição teórica dos modelos. Na segunda é apresentado o resultado dos modelos em conjunto com a apresentação dos próprios dados usados.

1 - Modelos

Nesta seção serão apresentados os modelos usados neste trabalho. O objetivo é apresentar o necessário para que o leitor entenda qual a finalidade e função de cada modelo dentro do trabalho. Portanto, essa seção não pretende ser uma revisão da literatura acerca dos modelos, muito menos esgotar a discussão sobre cada um deles.

Uma breve introdução ao tema das redes neurais será feita, mostrando em breve perspectiva histórica como sua teoria se desenvolveu e como tem sido sua aplicação. A partir disso, iremos introduzir os elementos que constituem uma rede neural e descrever, de forma sucinta, como esse tipo de modelo "aprende". Assim podemos apresentar, de fato, o modelo utilizado neste trabalho, que pertence a categoria das redes neurais recorrentes.

Feito isso, uma apresentação semelhante será feita para o modelo GARCH.

1.1 - Redes neurais artificiais

1.1.1 - O que é uma rede neural artificial?

Com os avanços da neurociência no século XX, não demoraria muito para que se tentasse formular matematicamente o funcionamento de um neurônio e de suas conexões. E em 1943, McCulloch e Pitts tentaram exatamente isso. A partir disso, é inaugurado o campo de estudo das redes neurais (RN) artificiais, porém a ideia de uma rede de funções matemáticas interligadas capazes de executar operações lógicas e computações pode ser muito mais antiga (SCHMIDHUBER,2015).

Muitos alterações no modelo de McCulloch e Pitts foram feitas para que chegássemos ao que entendemos hoje como uma RN artificial. Inicialmente, um neurônio é ativado na base do tudo ou nada, ou seja, ele está ativo ou não, possuindo assim uma característica binária, o que está de acordo com a biologia (MCCULLOCH;PITTS,1943). Hoje, as RN's tem em seu *output* valores que podem estar entre qualquer intervalo dos números reais, de acordo com a imagem de sua função de ativação.

Baseado na formulação proposta por McCulloch e Pitts, em 1957 Rosenblatt propõe o *perceptron* como um modelo artificial para o neurônio humano (ROSENBLATT,1957). Para entendermos o *perceptron*, imagine que temos um vetor X de tamanho n, cada elemento

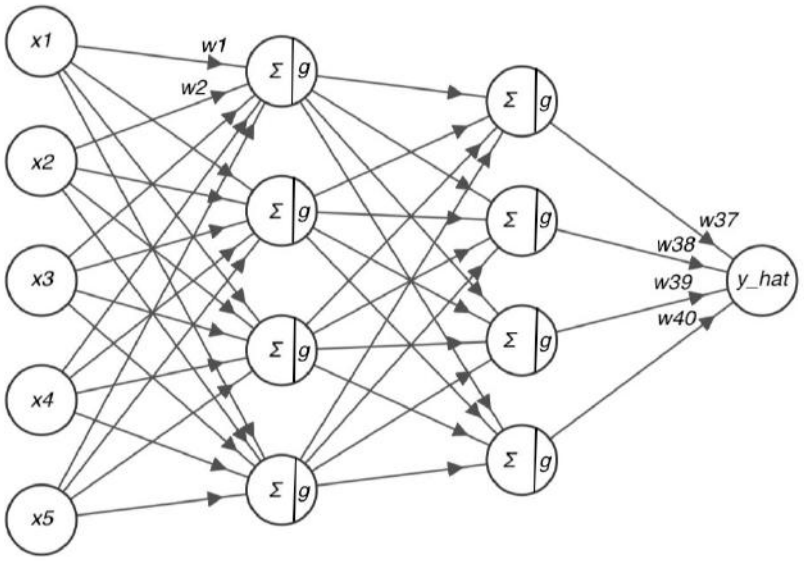
deste vetor descreve um atributo do *input* para o modelo. Dessa forma, somamos esses valores ponderando todos por um vetor W e adicionamos um escalar b, depois utilizamos esse resultado em uma função (ROSENBLATT,1957). Portanto, para um *perceptron* temos:

$$\widehat{y} = g(\sum_{i=1}^{n} (x_i * w_i) + b)$$

Em que x_i e w_i são os elementos dos respectivos vetores X e W, b é chamado de viés. O vetor W e o viés b são parâmetros do modelo. Essa função, $g(x)$, é chamada de função de ativação e pode assumir diversas formas. Em suas primeiras formulações apresentava, também, um *output* binário, em que o somatório deveria atingir um determinado valor para que o *output* fosse 1, caso contrário o resultado seria 0 (AURÉLIEN,2019).

Com o encadeamento de vários *perceptrons* em camadas, temos a primeira RN artificial, chamada de *multilayer perceptron* (MLP) (AURÉLIEN,2019). Nesse tipo de modelo, todos os neurônios de uma camada se conectam a todos os neurônios da próxima, formando uma camada densa (AURÉLIEN,2019). Uma das primeiras aplicações desse modelo foi a de classificação de letras do alfabeto a partir de estímulos visuais (ROSENBLATT,1960). Sua representação gráfica tem a forma:

Figura 1: multilayer perceptron



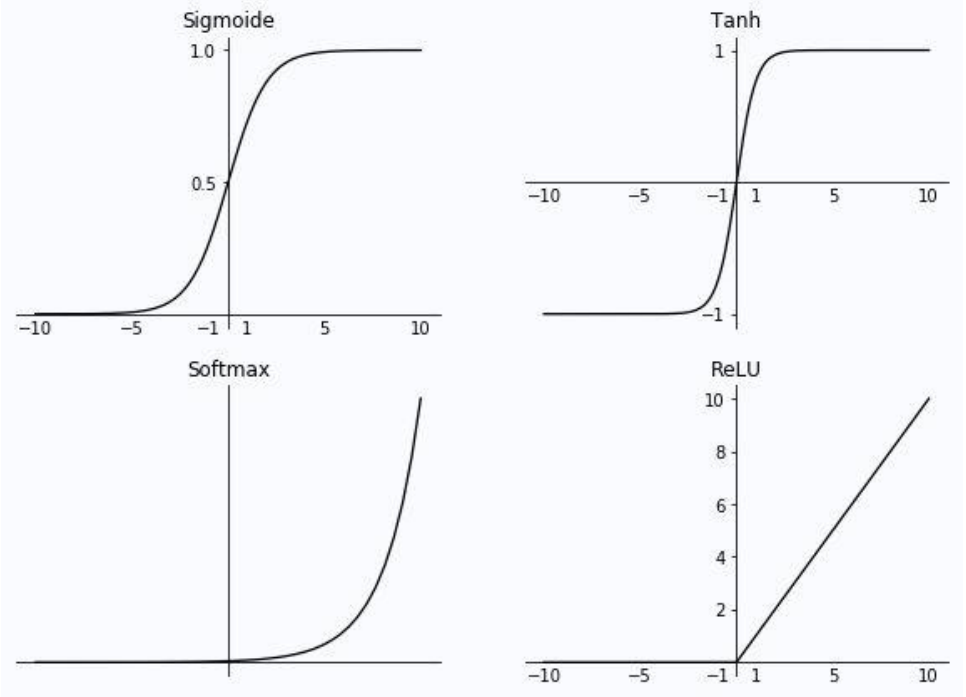
Fonte: Elaboração própria

Nessa representação, temos como *input* um vetor com n igual a 5, 2 camadas intermediárias com 4 *perceptrons* ou neurônios cada e um neurônio de *output*. O neurônio de

output final também é um *perceptron*, apenas foram omitidas suas funções internas. Também foram omitidos os vieses b para cada neurônio. Logo, temos 5 parâmetros w_i e um viés b para cada neurônio da primeira camada, 4 parâmetros w_i e um b para cada neurônio da segunda e, finalmente, 4 parâmetros w_i e um b para o *output*, totalizando 40 pesos w_i e 9 vieses b.

Atualmente, as funções de ativação mais utilizadas costumam ser a sigmóide, *softmax*, a tangente hiperbólica (tanh) e a retificadora linear unitária (ReLU) (AURÉLIEN,2019). Cada função deve ser escolhida de acordo com a tarefa a ser feita, sendo a sigmóide e *softmax* usadas, comumente, para classificação (AURÉLIEN,2019). Já a tanh e a ReLU podem ser usadas em diversas situações, incluindo classificação, porém apenas nas camadas intermediárias, deixando a última camada com uma função de ativação específica para classificação (AURÉLIEN,2019).

Gráfico 1: funções de ativação



Fonte: Elaboração própria

Então, resumindo essa subseção, um MLP é apenas uma sequência arbitrariamente longa de somatórios ponderados seguidos de uma função de ativação, em que o *output* de uma camada se torna o *input* para próxima. Uma outra definição interessante é que MLPs são aproximadores universais, capazes de aproximar qualquer função com um certo grau de

precisão, dado uma quantidade de neurônios e camadas (HORNIK;STINCHCOMBE;WHITE,1989).

RN artificiais podem ser entendidas como uma generalização de MLPs, não necessariamente fazem apenas somatórios dentro dos seus neurônios e, também, podem ter diversas topologias, ou seja, uma camada pode se conectar a uma próxima de várias formas, pulando algumas camadas, se retroalimentando, não se conectando a nenhuma, existem inúmeras possibilidades (AURÉLIEN,2019).

Porém, essas definições não mencionam um elemento fundamental desses tipos de modelos, que desempenha papel central no processo de "aprendizado".

1.1.2 - Como RN artificiais aprendem?

Uma das grandes dificuldades de utilizar MLPs ocorria em seu processo de treinamento ou aprendizado (AURÉLIEN,2019). Mas o que é, em termos simples, esse processo? A grosso modo, falando em RN artificiais, o processo de aprendizado é um processo em que se busca os parâmetros ideais do modelo para que se minimize a sua função de erro e uma dada tarefa seja executada (SCHMIDHUBER,2015). Essa função de erro é chamada de função de perda ou custo e, novamente, pode assumir diversas formas (AURÉLIEN,2019).

$$custo = L(Y, \widehat{Y})$$

Ela é computada tomando os *outputs* do modelo $\widehat{Y}$ e comparando com os resultados verdadeiros Y, sendo ambos vetores em que cada valor corresponde a "etiqueta" de uma observação usada como *input*. Assim, os MLPs são chamados de modelos de aprendizado supervisionado, pois existe a ideia de um "supervisor" que conhece as respostas certas e etiqueta os dados (AURÉLIEN,2019). Existem RN artificiais usadas em problemas não supervisionados, porém descrever o funcionamento de tais modelos foge ao escopo do trabalho.

Em 1985, pensando na dificuldade de se treinar MLPs, foi apresentado um algoritmo que hoje é o padrão quando se fala em treinamento de RN artificiais, generalizando para além dos MLPs (RUMELHART;HINTON;WILLIAMS,1985). Em sua aparição inicial foi chamado de regra delta generalizada mas, atualmente, é conhecido como retropropagação (AURÉLIEN,2019). Uma descrição formalmente precisa de tal processo está além das

pretensões desse trabalho mas a seguir é fornecida a sua intuição.

O processo ocorre em duas grandes etapas. O primeiro passo é a propagação para frente, em que os *inputs* percorrem toda a rede até que seu valores finais sejam computados ($\widehat{Y}$), e então é calculado o custo com a função de perda. A partir desse momento começa o processo de retropropagação, em que são tomadas as derivadas parciais da função de perda em relação aos parâmetros w_i e b, utilizando a regra da cadeia até se computar as derivadas parciais da camada mais próxima dos *inputs* (RUMELHART;HINTON;WILLIAMS,1985). Após computada todas as derivadas, é possível saber quanto cada parâmetro contribui para o erro. Com essa informação, os parâmetros são atualizados a fim de minimizar o erro, utilizando o processo de descida de gradiente (RUMELHART;HINTON;WILLIAMS,1985). A regra de atualização dos parâmetros é definida como:

$$w_i^{t+1} = w_i^t - \alpha \frac{\partial L}{\partial w_i^t}$$

Uma equação similar pode ser feita para os vieses b, α é chamado de taxa de aprendizagem. Após percorrer todo o conjunto de observações usadas como *input* e atualizar os parâmetros uma vez, é dito que um episódio ou *epoch* foi concluído (AURÉLIEN,2019). Quando o erro não consegue mais ser reduzido, ou já está dentro de uma margem aceitável, o modelo está treinado (SCHMIDHUBER,2015).

É importante destacar que para implementar o algoritmo de retropropagação é necessário que as funções de ativação e de perda tenham derivadas definidas, caso contrário não é possível propagar a computação das derivadas parciais através da regra da cadeia (AURÉLIEN,2019).

Um outro ponto importante é o valor inicial que os parâmetros devem ter para que o modelo possa ser treinado. Se todos os pesos w_i tiverem os mesmos valores, exatamente a mesma coisa será computada em todos os neurônios e todas as derivadas parciais serão iguais, impossibilitando assim que o modelo aprenda as diferentes relações entre *input* e as "etiquetas" (AURÉLIEN,2019). Portanto, é necessário quebrar a simetria do modelo inicializando os parâmetros com valores aleatórios. Existem inúmeras técnicas de inicialização de parâmetros, cada uma retirando os valores de uma distribuição específica com uma dada média e variância (AURÉLIEN,2019).

Uma técnica muito usada para acelerar o processo de treinamento e, portanto, reduzir o número de *epochs* necessários para treinar o modelo é chamada de *mini-batch*

(AURÉLIEN,2019). Essa técnica consiste em, ao invés de atualizar os parâmetros ao final de uma *epoch*, os parâmetros são atualizados após um certo número de observações (AURÉLIEN,2019). O tamanho da *mini-batch* pode estar entre 1 e o número de observações total do conjunto de dados, sendo esses chamados respectivamente de *batch* estocástico e *batch* (AURÉLIEN, 2019). A técnica de *mini-batch* é amplamente utilizada e apresenta resultados similares ao *batch*, porém com tempo de treinamento consideravelmente reduzido (SCHMIDHUBER,2015).

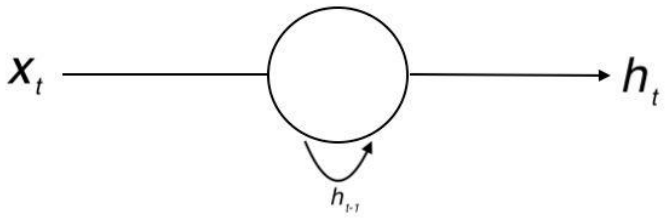
Para criar uma RN artificial deve-se então escolher o número de camadas, a quantidade de neurônios em cada uma delas, o tipo de neurônio, suas funções de ativação, a função de perda, a taxa de aprendizagem, o tamanho da *mini-batch*, a quantidade de *epochs* e a inicialização dos parâmetros. Apesar de não existir uma definição formal para o termo, esses atributos são chamados de hiperparâmetros, pois devem ser escolhidos antes de iniciar o treinamento do modelo, diferente dos parâmetros que são encontrados durante a aprendizagem (AURÉLIEN,2019). As possibilidades de criação de uma RN artificial não se esgotam apenas nessas escolhas, técnicas de regularização e outros algoritmos de otimização também podem ser escolhidos, só para citar algumas outras possibilidades.

1.1.3 - RN recorrentes

Os modelos apresentados até agora não possuem capacidade detectar relações temporais entre as observações. Para tarefas como classificação de imagens essa falta de capacidade não é relevante, porém quando o conjunto de observações é uma série temporal, ou seja, um conjunto de dados em que as observações possuem um componente sequencial e temporal, detectar tais relações passa a ter relevância no desempenho do modelo. Problemas envolvendo processamento de linguagem natural em áudio ou texto, composição de música, transcrição de áudio para texto e séries financeiras são exemplos de situações em que MLPs são pouco efetivas (HOCHREITER;SCHMIDHUBER,1997).

A solução para esse problema foram os neurônios recorrentes simples em que, além da computação típica de um *perceptron*, o mesmo possui uma memória interna capaz de armazenar, em algum nível, o *input* anterior e propagar sua informação para o próximo passo (KALMAN;KWASNY,1997). Essa memória é chamada de estado oculto da célula e o quanto será “lembrado” entre os passos é outro parâmetro a ser descoberto durante a aprendizagem.

Figura 2: neurônio recorrente

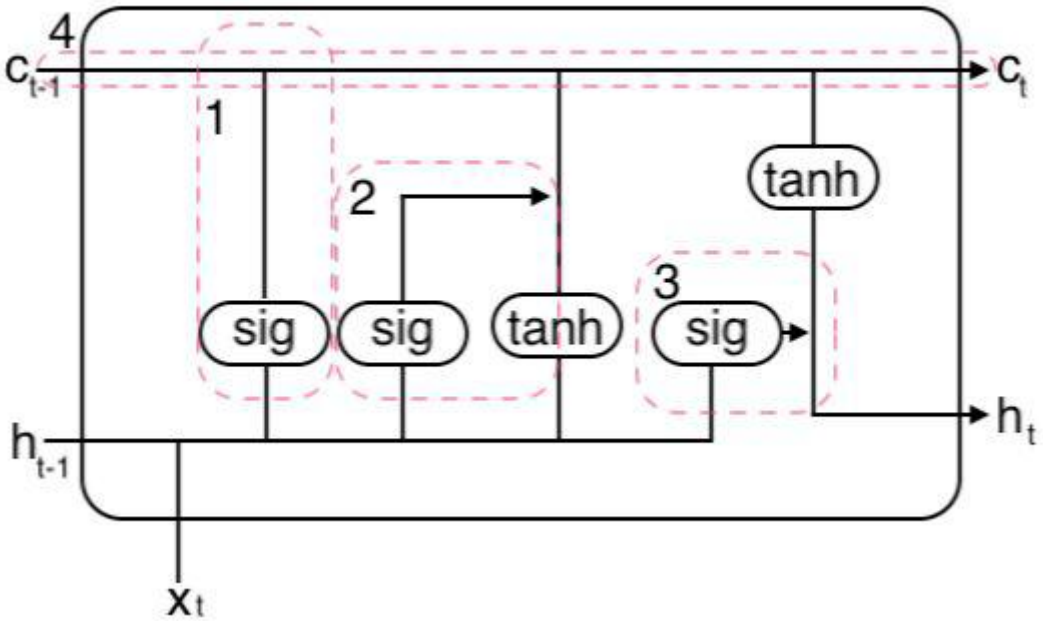


Fonte: Elaboração própria

Esse novo modelo, apesar de apresentar vantagens em relação ao MLP, possui duas falhas graves. Primeiro, ele sofre de memória curta, as observações feitas recentemente tem um peso muito maior para o seu *output* do que as mais antigas e, dado uma sequência longa o suficiente, o peso das primeiras observações pode ser praticamente nulo (HOCHREITER;SCHMIDHUBER,1997). Segundo, o processo de aprendizado pode sofrer de explosão/desaparecimento de gradiente, impedindo que a rede aprenda totalmente ou precise de um tempo proibitivo para ser treinada (HOCHREITER;SCHMIDHUBER,1997). Esse problema também é comum em RN artificiais com muitas camadas, chamadas de redes profundas (AURÉLIEN,2019). Essa falha ocorre durante a retropropagação, as derivadas parciais tomam valores gradativamente maiores/menores, a ponto da rede divergir da solução ótima (gradiente explodindo) ou não conseguir mais atualizar seus parâmetros (gradiente desaparecendo) (AURÉLIEN,2019).

Então, uma novo tipo de neurônio recorrente foi proposto, conhecido como *long short term memory* (LSTM) (HOCHREITER;SCHMIDHUBER,1997). Uma célula LSTM consegue lidar com ambos os problemas e, apesar da sua complexidade, possui tempo de treinamento não muito longo, tendo desempenho superior às técnicas anteriores em atividades de processamento de linguagem natural e outras tarefas com dados sequenciais (SCHMIDHUBER, 2015). A representação de um neurônio LSTM tem a forma:

Figura 3: célula LSTM



Fonte: Elaboração própria

Uma célula LSTM é composta de vários portões, regiões 1 a 3, e seu estado atual, região 4. A intuição de sua estrutura é que existem 2 memórias, uma longa e outra curta, e as funções sigmóide (sig) são utilizadas para controlar o quanto de informação irá ser mantida em cada uma delas (AURÉLIEN,2019). As funções tanh são utilizadas para comprimir os valores entre -1 e 1 e impedir a explosão de gradiente. A seguir uma explicação detalhada será feita.

Para entender o que cada região está fazendo deve-se compreender o que está sendo usado como *input*. Primeiro, x_t e h_{t-1} correspondem, respectivamente, ao *input* no tempo *t* e ao *output* no tempo *t-1*, que pode ser entendido também como a memória curta da célula (HOCHREITER;SCHMIDHUBER,1997). A região 4 é chamada de estado atual da célula com seu *input* c_{t-1}, que corresponde a memória longa. A região 1 então é chamada de "portão de esquecimento" em que é escolhido o quanto da memória longa c_{t-1} será mantida/descartada nessa etapa (HOCHREITER;SCHMIDHUBER,1997). A região 2 corresponde ao "portão de *input*" em que é decidido o quanto da memória curta h_{t-1} e do *input* x_t serão incorporados ao estado atual (HOCHREITER;SCHMIDHUBER,1997). A região 3 é o "portão de *output*" e combina o *input* ao estado da célula para gerar o *output* h_t,

porém sem alterar novamente seu estado c_t, que agora é propagado para a próxima etapa paralelamente ao *output* h_t (HOCHREITER;SCHMIDHUBER,1997).

Trazendo ao caso concreto de uma série financeira, o termo h_{t-1} corresponde a uma espécie de modelo AR(1). Já a memória longa c_{t-1} é um valor que codifica informações relevantes para o período, por exemplo, se a série atingiu algum topo ou fundo recentemente, se cruzou alguma banda de um indicador financeiro, etc. Nesses casos, o neurônio que capta essa informação específica ficaria estimulado indicando para a RN a ocorrência daquele padrão, mesmo depois de alguns períodos. Assim, cada neurônio LSTM pode codificar em sua memória longa uma informação diferente que seja importante para a previsão.

Agora já temos a base para introduzir os modelos de RN artificiais usados neste trabalho.

1.1.4 - Definição do modelo RN artificial

RN artificiais estão mais próximas da engenharia do que da ciência, no sentido que muitas vezes não existem regras embasadas na teoria para decidir pelos melhores hiperparâmetros, apenas regras práticas que dão resultados em alguns casos (SMITH,2018). Com isso, o melhor método para encontrar modelos que funcionem é a tentativa e erro. Algumas possibilidades foram tentadas ao longo do trabalho, algumas apresentando resultados superiores ao *benchmark* escolhido, discussão que será melhor apresentada na seção 2. Aqui apenas será exposto as arquiteturas escolhidas para serem treinadas.

As características mantidas constantes entre todos os modelos foram: o tamanho da *mini-batch* de 8 observações, a taxa de aprendizagem de 10^{-3}, o erro quadrado médio (MSE) como função de perda e a métrica usada para escolher os modelos foi a raiz do erro quadrado médio (RMSE). O número de *epochs* tem pouco relevância pois foi utilizada a técnica de "parada prematura", que consiste em encerrar o treinamento caso a função de perda não esteja mais sendo reduzida de forma significativa. Geralmente, o encerramento do treinamento acontecia entre 15 e 30 *epochs*. A técnica de inicialização dos parâmetros utilizada foi a padrão da biblioteca *TensorFlow/Keras.*

As arquiteturas treinadas variaram entre 1 a 3 camadas internas mais a camada de *output*. O número de células LSTM em cada camada interna variou de 110, 90, 70 e 50 neurônios cada. A camada de output sempre possuía apenas 1 neurônio simples e sua função

de ativação variou entre a tanh, exponencial e linear. Com isso então foram treinados 36 modelos diferentes usando apenas os retornos da série como *input*, 8 dessas arquiteturas foram escolhidas e treinadas novamente usando também a volatilidade fornecida pelos modelos GARCH escolhidos. Todos os resultados serão apresentados na seção 2.

1.2 - Modelo GARCH

RN artificiais são modelos que buscam executar uma tarefa, seja ela de regressão, classificação ou previsão, sem ter preocupação em explicar o funcionamento do processo que gerou os dados em primeiro lugar. Modelos econométricos, apesar de serem usados também para previsão, tentam modelar os dados ali presentes. Dentro desse paradigma, explicar o funcionamento da volatilidade em séries financeiras é uma grande questão para economistas e financistas, dado a sua importância para a precificação de ativos financeiros variados e para questões de planejamento e estratégia (POON;GRANGER,2003).

Com isso, em 1982 foi proposto por Engle o modelo ARCH e, posteriormente, em 1986 foi feita a sua generalização por Bollerslev, dando início a criação da família de modelos GARCH. Diferentes dos modelos de séries temporais que existiam anteriormente, que pressupõem a volatilidade como constante ao longo do tempo, os modelos ARCH/GARCH tornam explícito a diferença entre a variância condicional e incondicional (BOLLERSLEV,1986). A partir disso, é modelada a variância condicional em função do conjunto de informações existentes até o momento, ou seja, o passado condiciona a volatilidade futura (BOLLERSLEV,1986). Os termos variância e volatilidade estão sendo como sinônimos neste trabalho.

Como o modelo ARCH é um caso especial modelo GARCH, sigla para *generalized autoregressive conditional heteroskedasticity* (em português: heteroscedasticidade condicional auto-regressiva generalizada), portanto apenas esta será apresentada e formulada. O modelo ARCH(p) é igual a um GARCH(0,p).

Uma série ε_t é dita como realização de processo GARCH(q, p) caso:

$$\varepsilon_t = \omega_t \sqrt{h_t}$$

$$h_t = \alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j h_{t-j}$$

O termo ω_t é a realização de um processo de ruído branco com média zero e variância igual a 1. Na equação de h_t, o primeiro termo é uma constante. O segundo termo é o elemento auto regressivo da série, sendo p o parâmetro que especifica quantas observações passadas devem ser incorporadas ao elemento atual da série (COWPERTWAIT;METCALFE,2009). Já o terceiro termo incorpora a variância condicional anterior, dando um caráter similar a um processo de MA (médias móveis) usado em modelos ARIMA, sendo q o seu parâmetro (BOLLERSLEV,1986). Porém, diferente de um processo ARIMA, o elemento de MA está sendo incorporado a variância e não a média. Logo, é possível demonstrar para um processo GARCH(1,1):

$$var(\varepsilon_t) = E(\varepsilon_t^2)$$

$$var(\varepsilon_t) = E(\omega_t^2)E(h_t)$$

$$var(\varepsilon_t) = E(\alpha_0 + \alpha_1 \varepsilon_{t-1}^2 + \beta_1 h_{t-1})$$

$$var(\varepsilon_t) = \alpha_0 + \alpha_1 E(\varepsilon_{t-1}^2) + \beta_1 E(h_{t-1})$$

$$var(\varepsilon_t) = \alpha_0 + \alpha_1 var(\varepsilon_{t-1}) + \beta_1 E(h_{t-1})$$

Sendo $var(x)$ a variância condicional e $E(x)$ seu valor esperado. É possível observar o sentido do termo heterocedástico, pois na equação acima temos que a variância é uma função da variância anterior e da esperança do termo h anterior, deixando claro que a variância não é uma constante.

Todo o processo de escolha dos parâmetros q e p, assim como a estimação dos coeficientes α e β, serão feitas na próxima seção do trabalho após a apresentação dos dados usados.

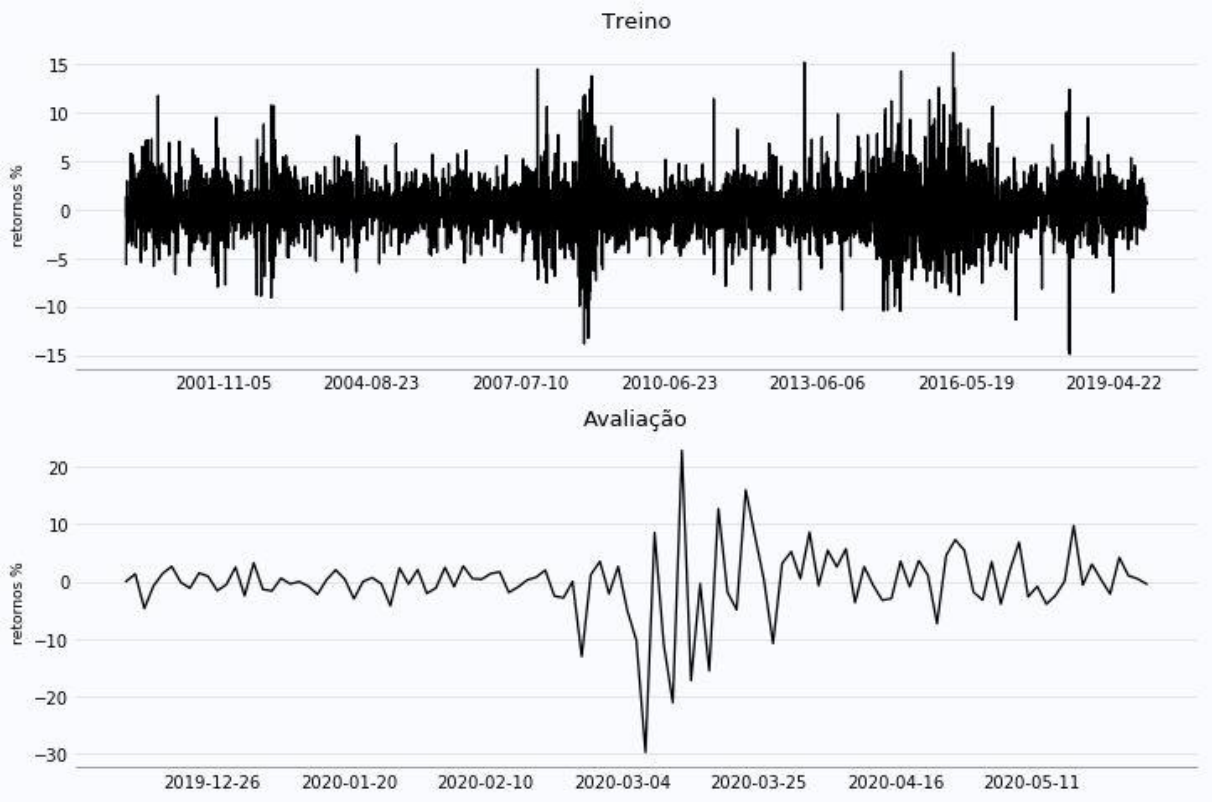
2 - Resultados

Agora, superado a parte teórica do trabalho, vamos introduzir a parte prática. Nesta seção serão apresentados os dados usados, a estimação dos modelos GARCH e a escolha dos parâmetros, a estimação das redes neurais e avaliação da sua capacidade preditiva. Após apresentado todos os resultados desses processos será feita uma consideração sobre a metodologia adotada.

2.1 - Dados

Os dados usados foram os retornos percentuais diários da ação PETR3, adquiridos através do site *yahoo! finance*. A plataforma fornece os preços de abertura, encerramento, mínimo e máximo diários. Com os preços de encerramento diários foram calculados os retornos percentuais. Os dados completos possuem 5134 observações e vão do dia 04/01/2000 até 28/05/2020. Foram separadas as primeiras 5021 observações para treinar os modelos e as últimas 113 observações foram usadas para avaliar a capacidade preditiva dos modelos.

Gráfico 2: retornos percentuais PETR3



Fonte: Elaboração própria

2.2 - Estimando GARCH

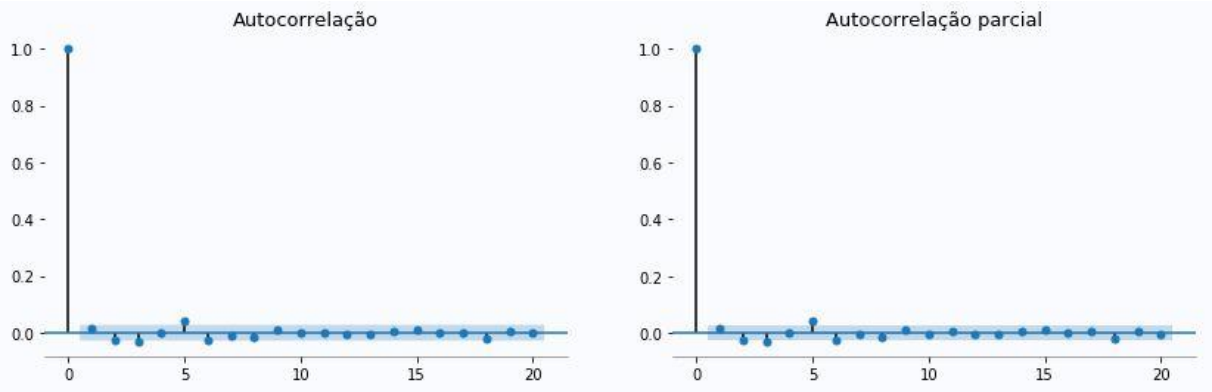
Antes de começar a escolha dos parâmetros e a estimação dos coeficientes é necessário verificar a presença de raiz unitária nos dados, o que indicaria que a série não é estacionária e, portanto, não pode ser estimada por um modelo GARCH[1]. Dessa maneira, foi

[1] Para estimar os parâmetros do modelo GARCH foi usada a biblioteca *ARCH* em sua versão para *Python* 3.x.

usado o teste de Dickey-Fuller aumentado, em que a hipótese nula é a presença de raiz unitária na série (STATSMODELS, 2019). Com nível de significância igual a 5%, deve-se rejeitar a hipótese nula de haver raiz unitária na série. Assim, é possível prosseguir para a escolha dos parâmetros.

A seguir serão apresentados os gráficos de autocorrelação[2] e autocorrelação parcial para os retornos.

Gráfico 3: autocorrelações

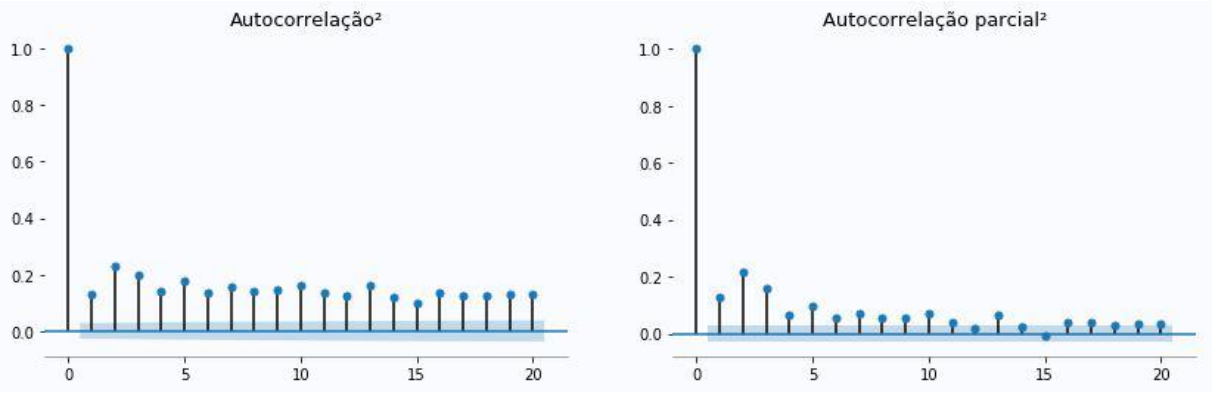


Fonte: Elaboração própria

É possível observar que não existe em ambos os gráficos autocorrelação significativa. Logo, a utilização de um modelo do tipo ARIMA para estimar essa série não parece ser uma escolha válida. Apenas o *lag* 5 está levemente fora da região de significância em 5%, o que poderia indicar autocorrelação para esse *lag*. Porém, como a preocupação principal de um modelo GARCH costuma ser a previsão da volatilidade, deve-se então olhar para os gráficos de autocorrelação da série ao quadrado. Tal técnica permite detectar a presença de autocorrelação na variância. Então:

[2] Para calcular as autocorrelações e outros testes estatísticos foi usada a biblioteca *Statsmodels* em sua versão para *Python* 3.x.

Gráfico 4: autocorrelações da série ao quadrado



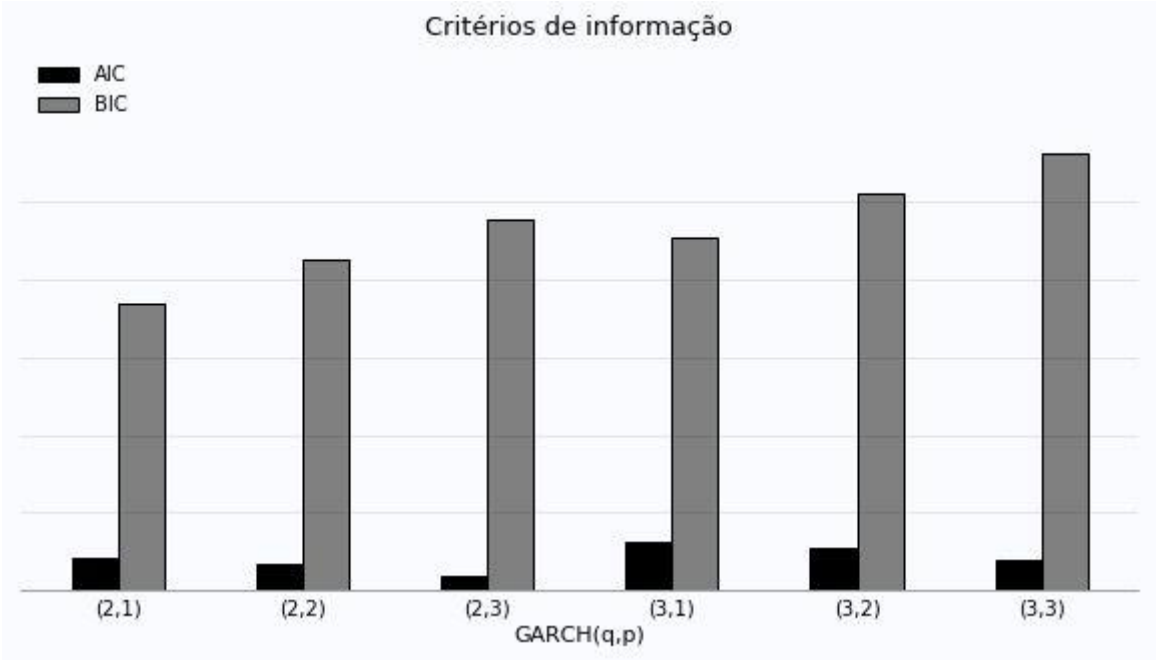
Fonte: Elaboração própria

A partir disso então temos um indicativo de uma possível autocorrelação na volatilidade. Assim, foram estimados vários coeficientes para diferentes combinações de parâmetros. Foram tentados todos os pares de p e q variando p entre 1 a 4 e q entre 0 a 4, ao todo foram 20 modelos.

Após estimados os coeficientes é necessário verificar se o modelo representa bem os dados. Uma forma de efetuar esta avaliação é o teste Ljung-box, em que se verifica se os resíduos do modelo são um ruído branco, caso essa hipótese seja a mais provável então o modelo está bem estimado para o conjunto de dados. A intuição do teste é que se os resíduos são apenas ruído então os coeficientes estão conseguindo explicar de forma satisfatória os dados e, portanto, não é necessário aumentar a quantidade de parâmetros. Nesse teste, a hipótese nula é a ausência de autocorrelação nos resíduos (STATSMODELS, 2019). O mesmo teste também será feito para os resíduos ao quadrado e o nível de significância considerado foi de 5%.

A seguir serão mostrados os critérios de informação AIC e BIC para os modelos que não apresentaram autocorrelação nos resíduos e nos seus quadrados de acordo com o teste Ljung-box para *lag* até 20. Os resultados das estimações dos modelos escolhidos podem ser encontrados no anexo I.

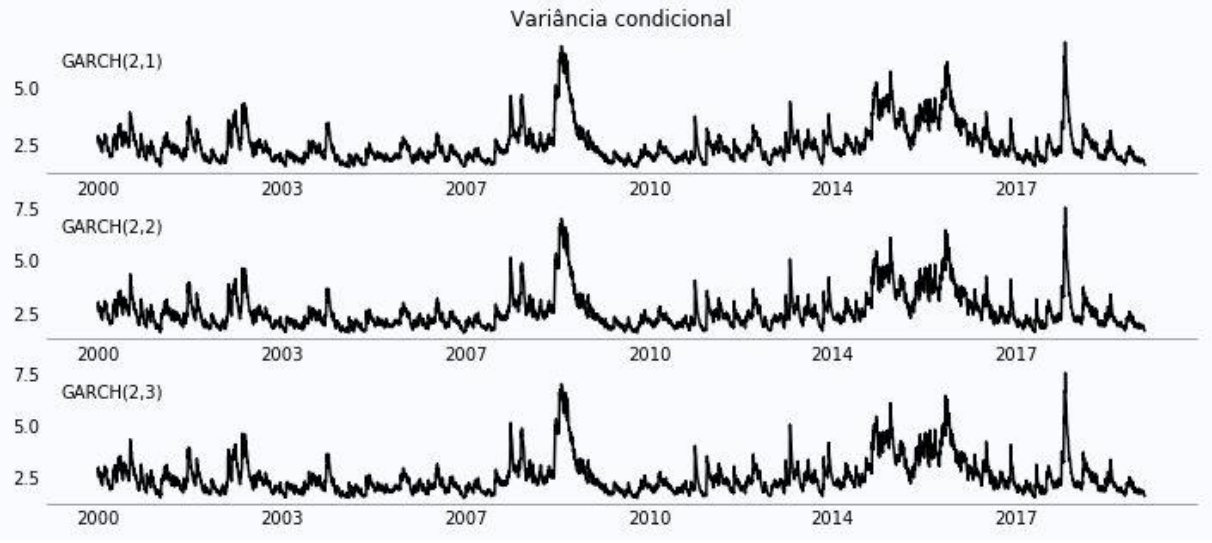
Gráfico 5: critérios de informação para os modelos GARCH



Fonte: Elaboração própria

De acordo com o critério AIC os 3 melhores modelos são os com parâmetros (2,3), (2,2) e (3,3). Para o critério BIC os 3 melhores são (2,1), (2,2) e (3,1). Então foram escolhidos os modelos (2,1), (2,2) e (2,3) por estarem todos com bons resultados em ambos os critérios. Esses modelos fornecerão a variância condicional como *input* para as redes neurais.

Gráfico 6: variâncias condicionais dos modelos GARCH escolhidos



Fonte: Elaboração própria

2.3 - Estimando as redes neurais

2.3.1 - Pré-processamento dos dados

Para treinar uma rede neural com dados sequências é necessário dividir os dados em janelas de *input* e janelas de *output*. O que está sendo feito na prática é dividir a série em uma sequência de vetores e, dessa forma, o modelo irá mapear cada um desses vetores a uma etiqueta. Essa etiqueta pode ser outro vetor ou um valor específico.

Os dados foram divididos em janelas de 21 observações com a observação número 22 correspondente a etiqueta, ou seja, está sendo usado um período de 21 dias para se prever o retorno do dia imediatamente após esse período. É completamente possível fazer previsões para períodos mais longos mas neste trabalho optou-se por esse tipo de previsão.

Quando for adicionado a variância condicional dos modelos GARCH o mesmo processo será feito. A etiqueta continuará sendo apenas o retorno no dia 22, logo o modelo estará mapeado uma matriz para um escalar.

Da perspectiva da RN, essas janelas são equivalentes a observações independentes. Portanto, a memória longa e curta da célula codifica eventos relevantes dentro desse período de 21 dias. Quando uma nova janela é fornecida ao modelo sua memória é "apagada", restando apenas a "estrutura" da memória, ou seja, quais padrões o neurônio em questão deve captar dentro no período.

2.3.2 - Benchmark

Para compararmos a capacidade preditiva dos modelos foi escolhida como métrica a RMSE. Porém, para uma comparação com capacidade interpretativa deve-se ter a RMSE de um modelo mais simples usado como *benchmark*. Para este trabalho foi escolhido usar a média histórica da série como *benchmark*. A média foi calculada retirando as últimas 113 observações por se tratarem do período para o qual será feita as previsões. Para esse período o *benchmark* teve um RMSE de 0.006768078.

Também usaremos a previsão da média dos modelos GARCH para comparação. Cada modelo teve o seguinte erro para o período: GARCH(2,1) com RMSE de 0.041094903,

GARCH(2,2) com RMSE de 0.04108905 e GARCH(2,3) com RMSE de 0.041089028. Logo, foi escolhido como referência o erro do GARCH(2,3) pois é o melhor resultado.

2.3.3 - Critérios de escolha entre modelos

Durante o processo de treinamento dos modelos é importante termos métricas para avaliar se o modelo está sofrendo de *overfitting*. Esse problema ocorre quando o modelo consegue mapear muito bem os dados para as etiquetas porém tem dificuldade de generalizar suas previsões para novos dados. Assim, é necessário ter uma parte dos dados separadas para fazer essa avaliação. Com 5021 observações para treinar o modelo foi separado 5% dessas observações para diagnosticar o *overfitting* e, dessa maneira, tem-se respectivamente o conjunto de treinamento e o conjunto de desenvolvimento. As 113 observações restantes correspondem ao conjunto de teste e servem ao propósito de auxiliar na escolha entre os modelos.

Apesar das funções MSE e RMSE serem muito parecidas matematicamente, cada uma serve um propósito diferente. A função de perda (MSE) é a referência usada durante o treinamento do modelo, portanto, ela é a função que será diferenciada e através da descida de gradiente será minimizada. Já RMSE não é diferenciada e, de forma similar ao conjunto de desenvolvimento, é usada para diagnosticar o *overfitting*. Esse diagnóstico é feito olhando para a RMSE no conjunto de treinamento e de desenvolvimento, caso o erro seja muito menor no primeiro caso é possível que o modelo esteja sofrendo de *overfitting*.

Além disso, essa mesma métrica (RMSE) será aplicada para as previsões feitas para o conjunto de teste. Os modelos escolhidos serão aqueles que apresentarem o menor RMSE para esse conjunto e que não estejam sofrendo de um nível alto de *overfitting*. Também foram escolhidos modelos que apresentavam arquiteturas com menor similaridade.

2.3.4 - Resultados dos modelos

A seguir é apresentada a tabela com as RMSEs[3] do conjunto de treino e para o conjunto de desenvolvimento para todos os modelos:

[3] Para criar os modelos foi usado a biblioteca *Tensorflow2/Keras* em sua versão para *Python* 3.x.

Tabela 1: erros para o conjunto de treino e de desenvolvimento

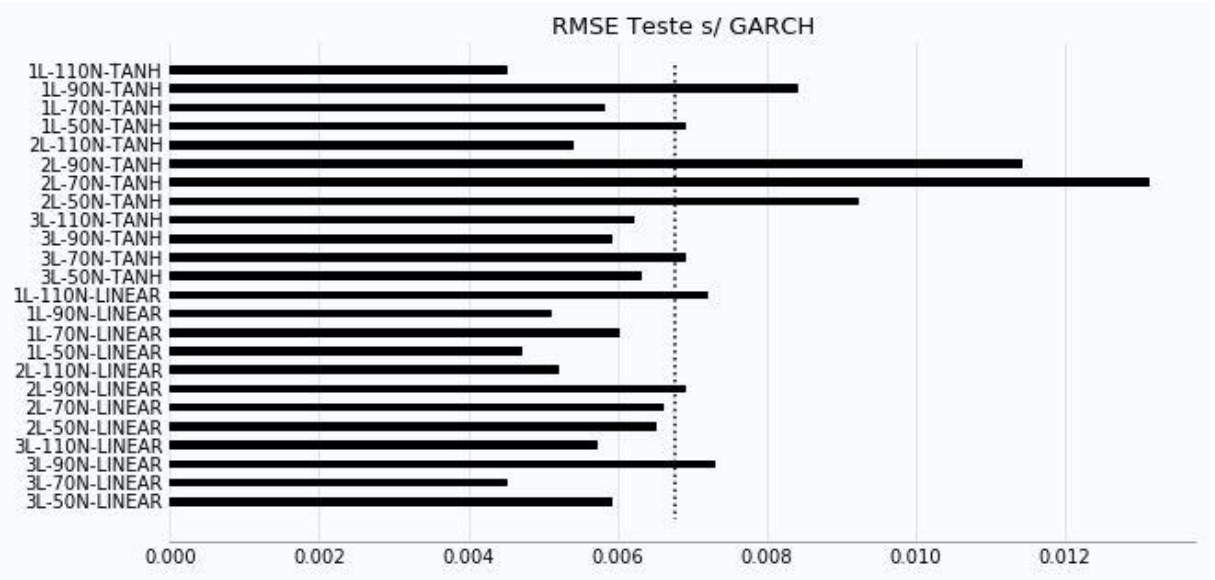
	RMSE Treino	RMSE Desenvolvimento
1L-110N-TANH	0.1724	0.1183
1L-90N-TANH	0.1723	0.1181
1L-70N-TANH	0.1723	0.118
1L-50N-TANH	0.1725	0.1179
2L-110N-TANH	0.1721	0.1184
2L-90N-TANH	0.1726	0.1179
2L-70N-TANH	0.1724	0.1182
2L-50N-TANH	0.1725	0.1181
3L-110N-TANH	0.1718	0.1191
3L-90N-TANH	0.1726	0.1178
3L-70N-TANH	0.1725	0.1178
3L-50N-TANH	0.1725	0.1177
1L-110N-LINEAR	0.1723	0.1187
1L-90N-LINEAR	0.1724	0.1182
1L-70N-LINEAR	0.1721	0.1182
1L-50N-LINEAR	0.1724	0.118
2L-110N-LINEAR	0.1722	0.1187
2L-90N-LINEAR	0.1723	0.1188
2L-70N-LINEAR	0.1724	0.1181
2L-50N-LINEAR	0.1723	0.1184
3L-110N-LINEAR	0.1726	0.1179
3L-90N-LINEAR	0.1725	0.1177
3L-70N-LINEAR	0.1725	0.1181
3L-50N-LINEAR	0.1725	0.1177
1L-110N-EXP	0.1758	0.123
1L-90N-EXP	0.1758	0.123
1L-70N-EXP	0.1758	0.123
1L-50N-EXP	0.1758	0.123
2L-110N-EXP	0.1758	0.123
2L-90N-EXP	0.1758	0.123
2L-70N-EXP	0.1758	0.123
2L-50N-EXP	0.1758	0.123
3L-110N-EXP	0.1758	0.123
3L-90N-EXP	0.1758	0.123
3L-70N-EXP	0.1758	0.123
3L-50N-EXP	0.1758	0.123

Fonte: Elaboração própria

Em geral, os erros não foram muito diferentes entre si, com o erro no conjunto de desenvolvimento sempre menor do que no conjunto de treino, o que não é comum. Uma possibilidade para explicar tal fenômeno é que o conjunto de desenvolvimento é muito mais curto que o de treino, logo, menos erros poderiam ser cometidos.

Também é possível perceber uma anomalia nos modelos que usavam a função exponencial na ativação de sua camada de *output*. Nestes modelos, os erros foram todos exatamente iguais, o que indica uma falha durante o processo de aprendizagem. Portanto, esses modelos serão todos descartados para as próximas análises. A seguir são mostrados os erros para o conjunto de teste:

Gráfico 7: erros para o conjunto de teste



Fonte: Elaboração própria

No gráfico, a letra "L" indica quantas camadas intermediárias o modelo possui, o "N" indica quantos neurônios estão presentes em cada camada e o texto final indica qual a função de ativação usada. A linha tracejada indica o erro do *benchmark*. O erro do GARCH não foi apresentado pois está muito fora da escala, ou seja, todos os modelos tem um erro menor que o GARCH.

Então foram escolhidos 4 modelos com a função de ativação tanh e mais 4 modelos com a função de ativação linear. Os modelos escolhidos foram: 1L-110N-TANH, 1L-70N-TANH, 2L-110N-TANH, 3L-90N-TANH, 1L-90N-LINEAR, 1L-50N-LINEAR, 2L-110N-LINEAR e 3L-70N-LINEAR. Todos apresentaram um RMSE inferior ao *benchmark* para o conjunto de teste.

Com essas 8 arquiteturas, iremos treinar novamente os modelos usando agora as variâncias condicionais fornecidas pelos modelos GARCH. Serão usadas as variâncias dos 3 modelos simultaneamente, além dos retornos. A ideia é permitir que a rede neural escolha qual série das variâncias é mais relevante para prever o retorno. A seguir é apresentado os erros dos modelos treinados com o GARCH para o conjunto de treino e desenvolvimento:

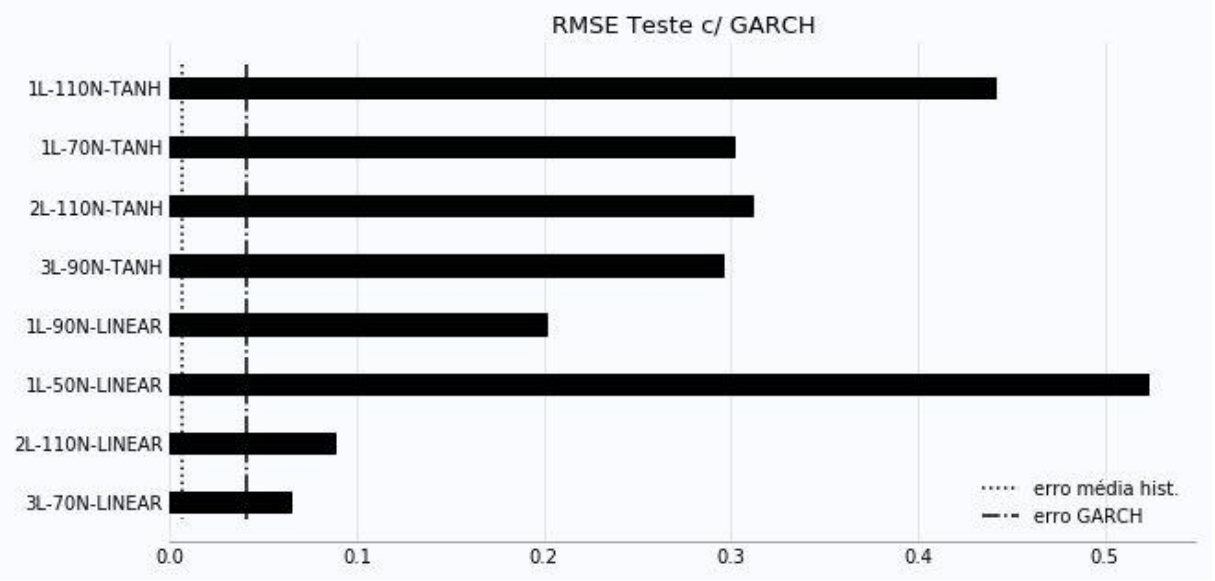
Tabela 2: erros para o conjunto de treino e desenvolvimento para os modelos com GARCH

	RMSE Treino	RMSE Desenvolvimento
1L-110N-TANH	0.1723	0.1176
1L-70N-TANH	0.1723	0.1176
2L-110N-TANH	0.1724	0.1177
3L-90N-TANH	0.1726	0.1176
1L-90N-LINEAR	0.1724	0.1176
1L-50N-LINEAR	0.1724	0.1175
2L-110N-LINEAR	0.1723	0.1176
3L-70N-LINEAR	0.1724	0.1176

Fonte: Elaboração própria

Para o conjunto de teste temos o seguinte gráfico:

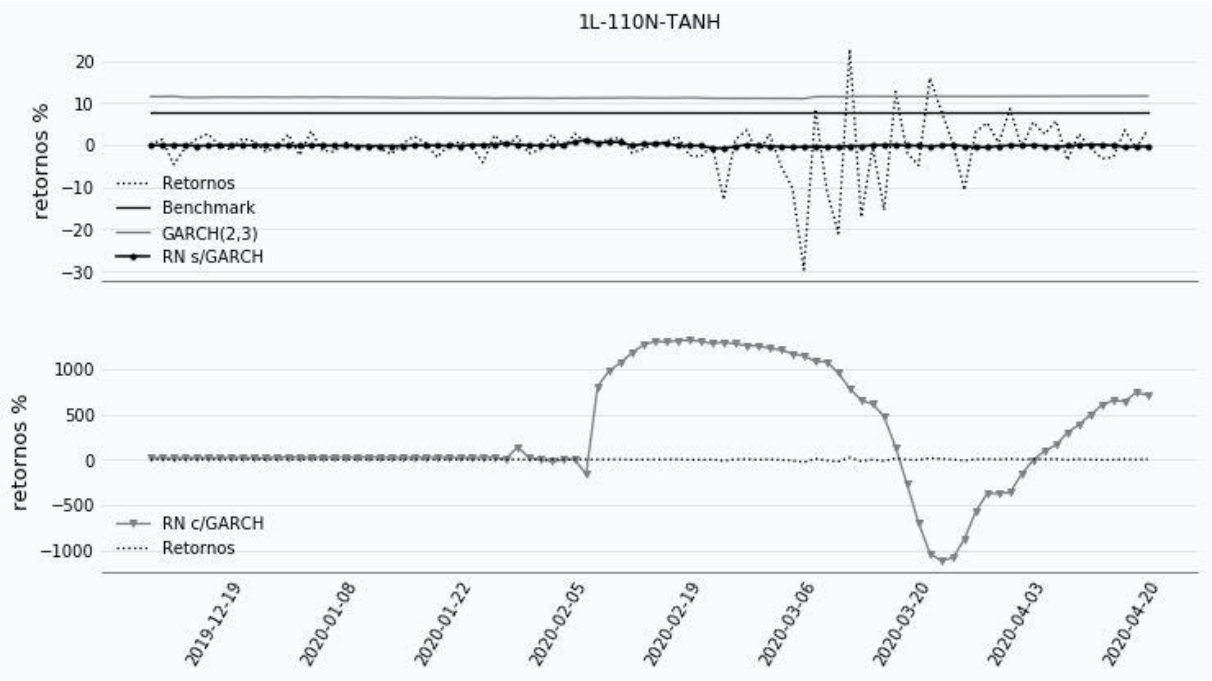
Gráfico 8: erros para o conjunto de teste para os modelos com GARCH



Fonte: Elaboração própria

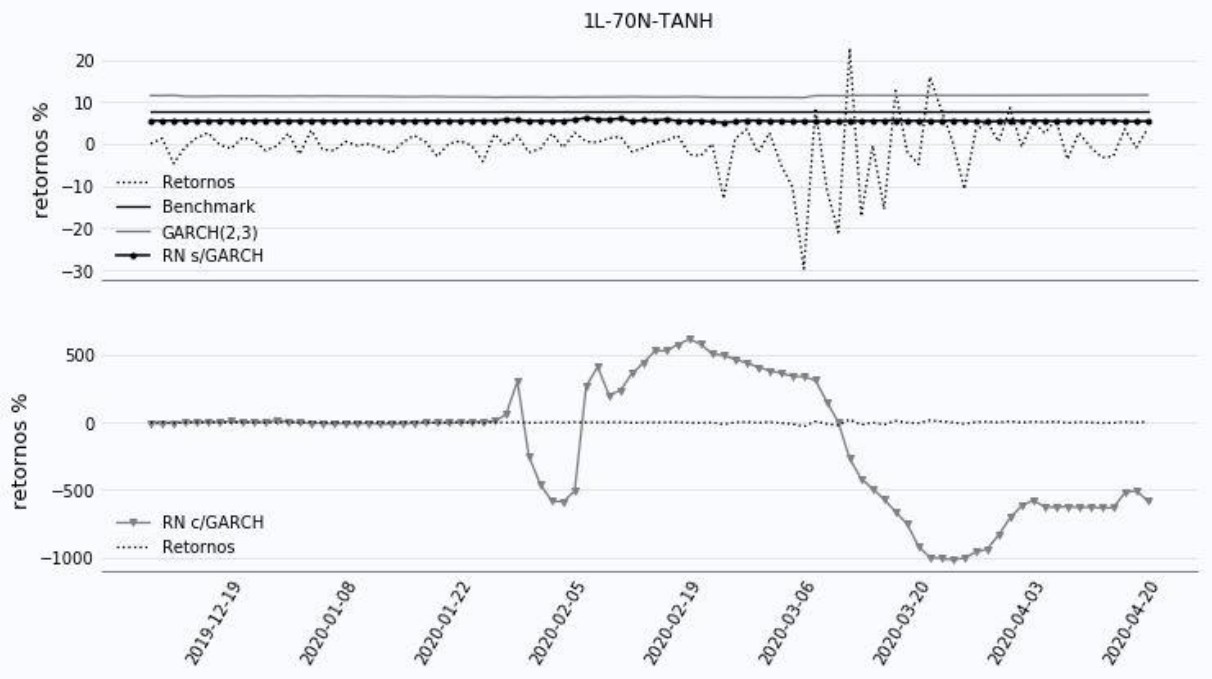
É possível afirmar que o desempenho dos modelos foi consideravelmente reduzido com a introdução da variância fornecida pelos GARCHs. Nem mesmo houve um desempenho superior ao próprio GARCH. A seguir é apresentado as previsões feitas para o período de teste para cada um dos 8 modelos.

Gráfico 9: retornos modelo 1L-110N-TANH



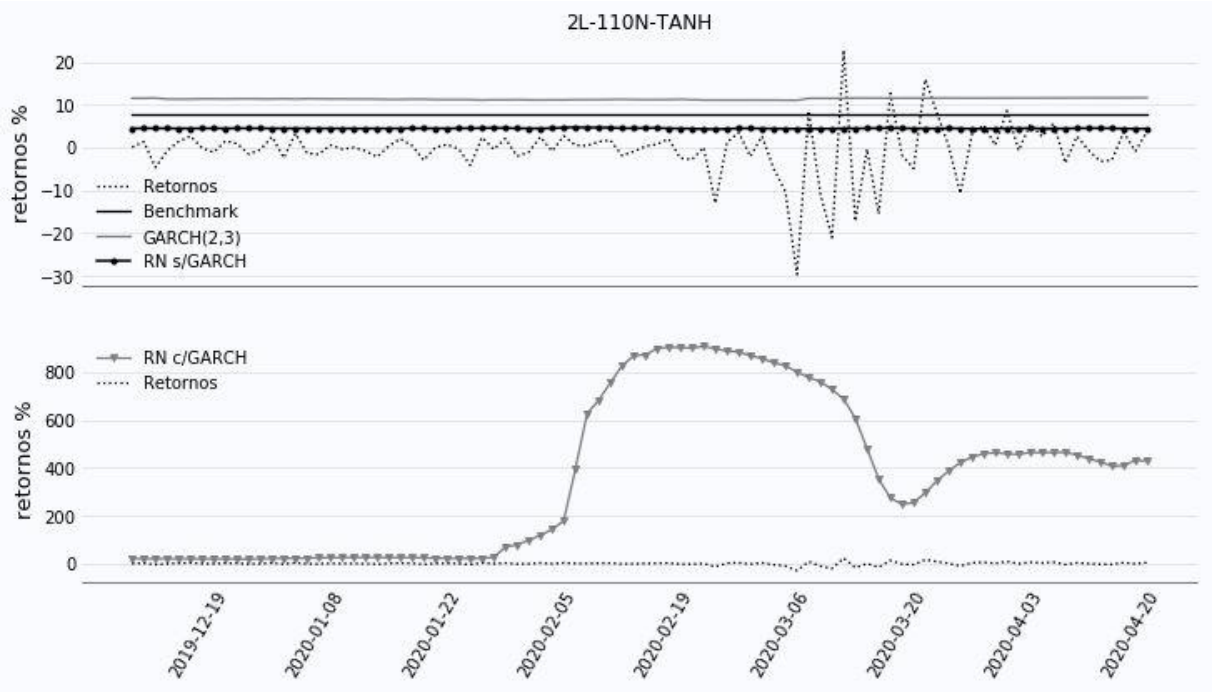
Fonte: Elaboração própria

Gráfico 10: retornos modelo 1L-70N-TANH



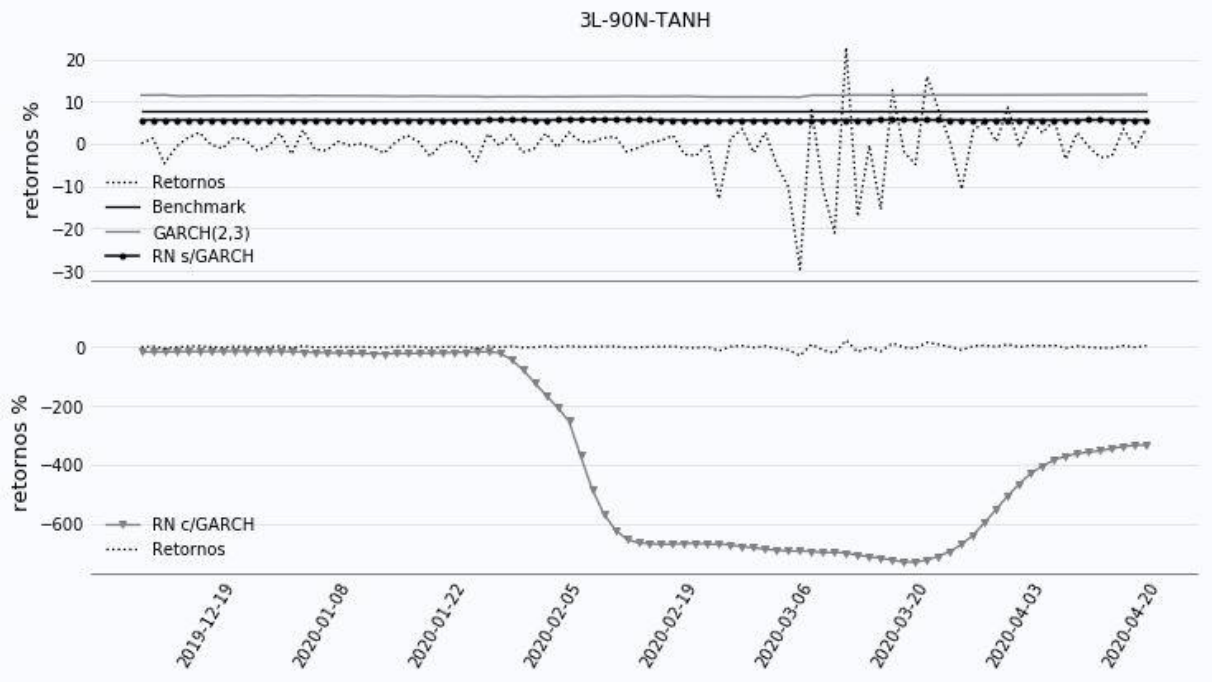
Fonte: Elaboração própria

Gráfico 11: retornos modelo 2L-110N-TANH



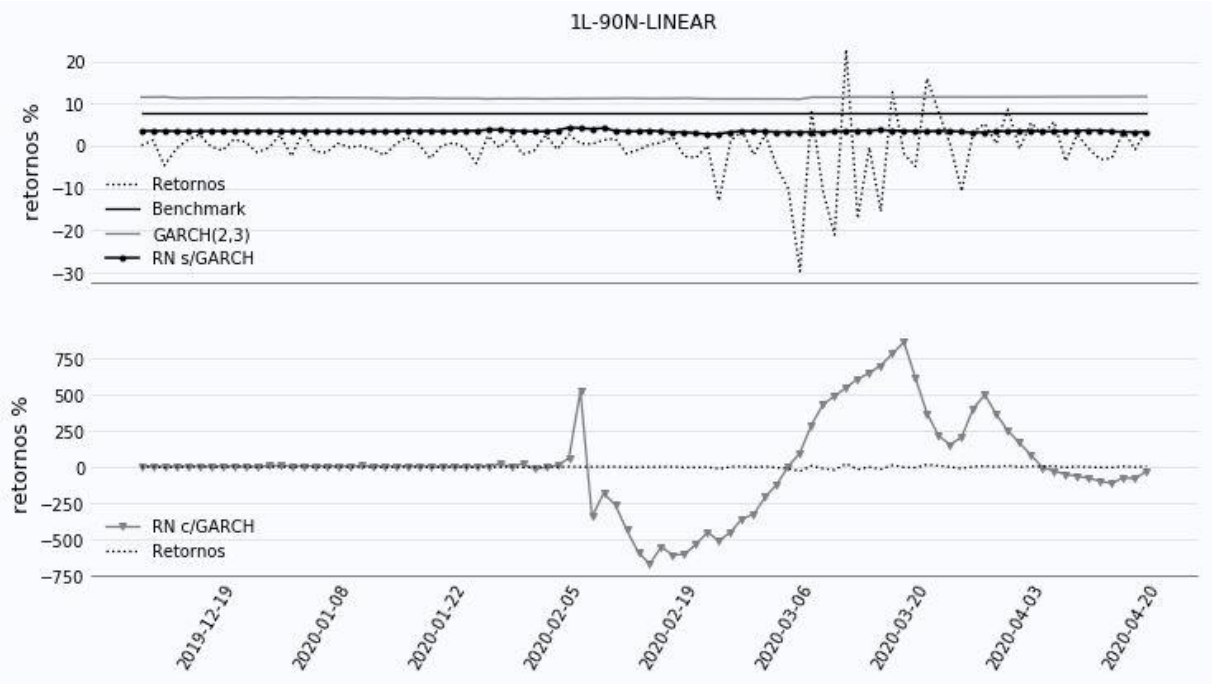
Fonte: Elaboração própria

Gráfico 12: retornos modelo 3L-90N-TANH



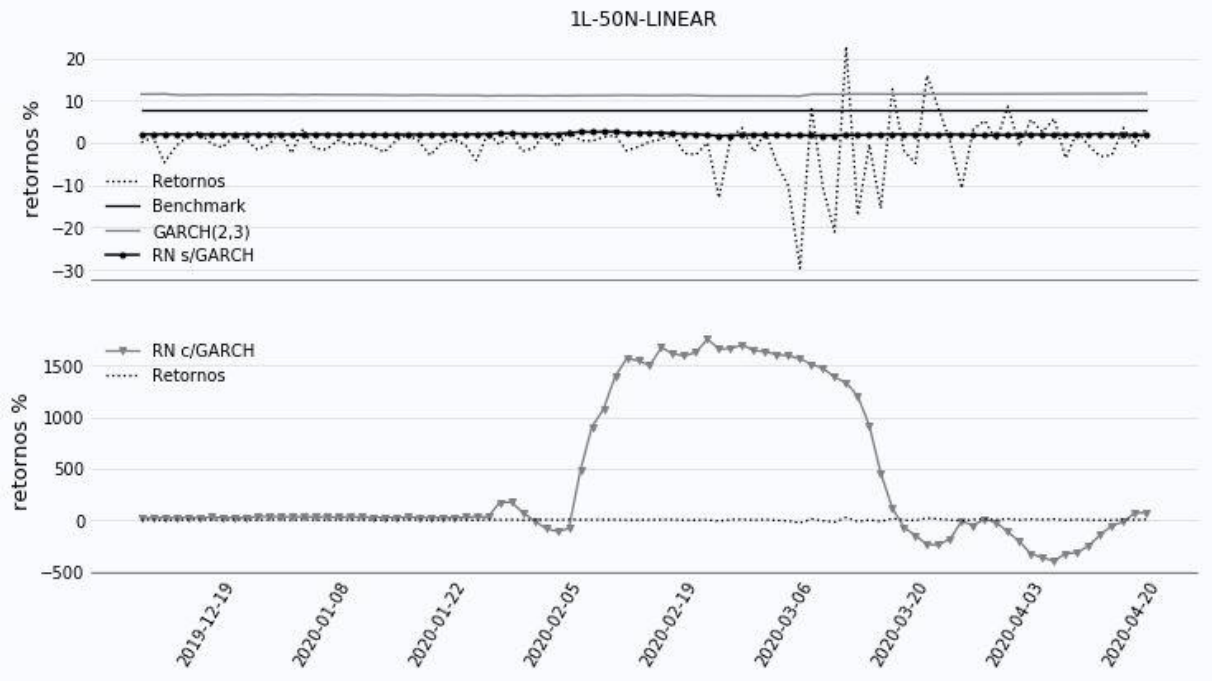
Fonte: Elaboração própria

Gráfico 13: retornos modelo 1L-90N-LINEAR



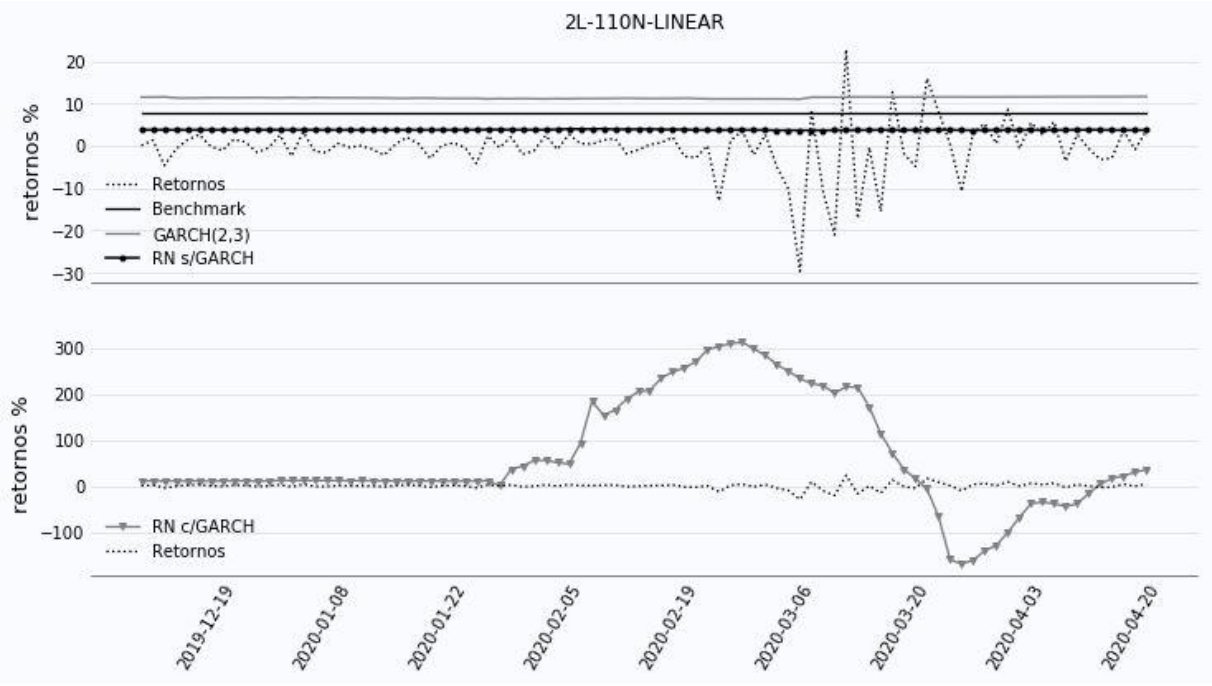
Fonte: Elaboração própria

Gráfico 14: retornos modelo 1L-50N-LINEAR



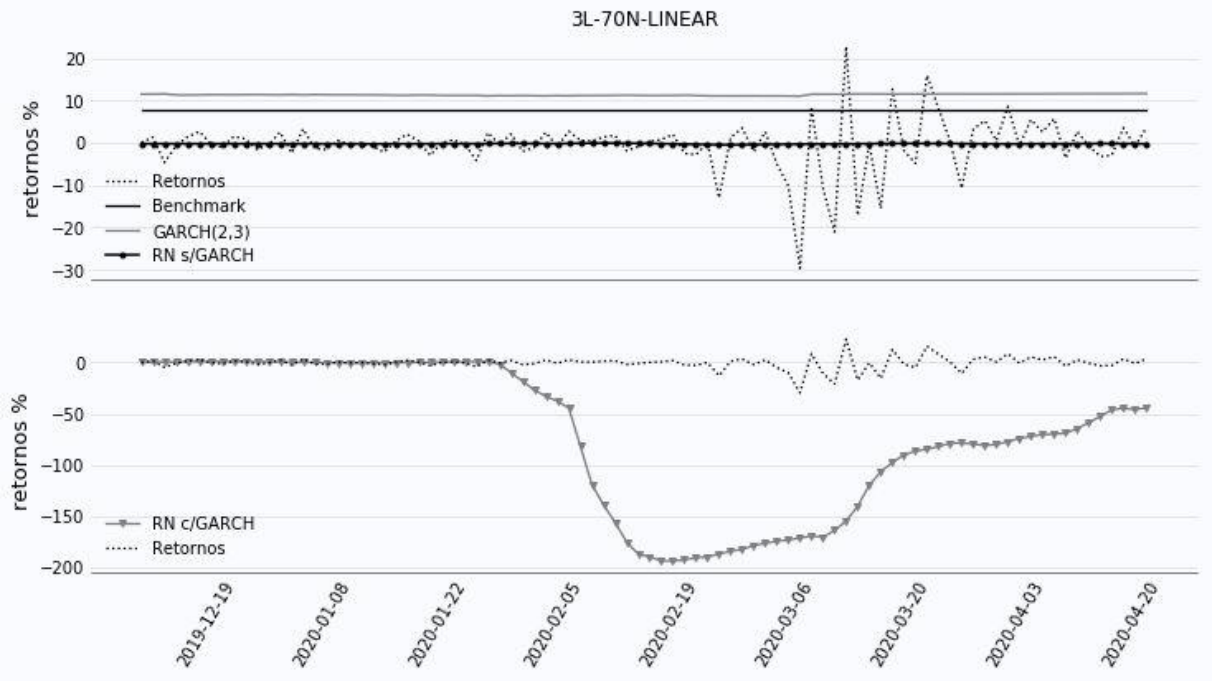
Fonte: Elaboração própria

Gráfico 15: retornos modelo 2L-110N-LINEAR



Fonte: Elaboração própria

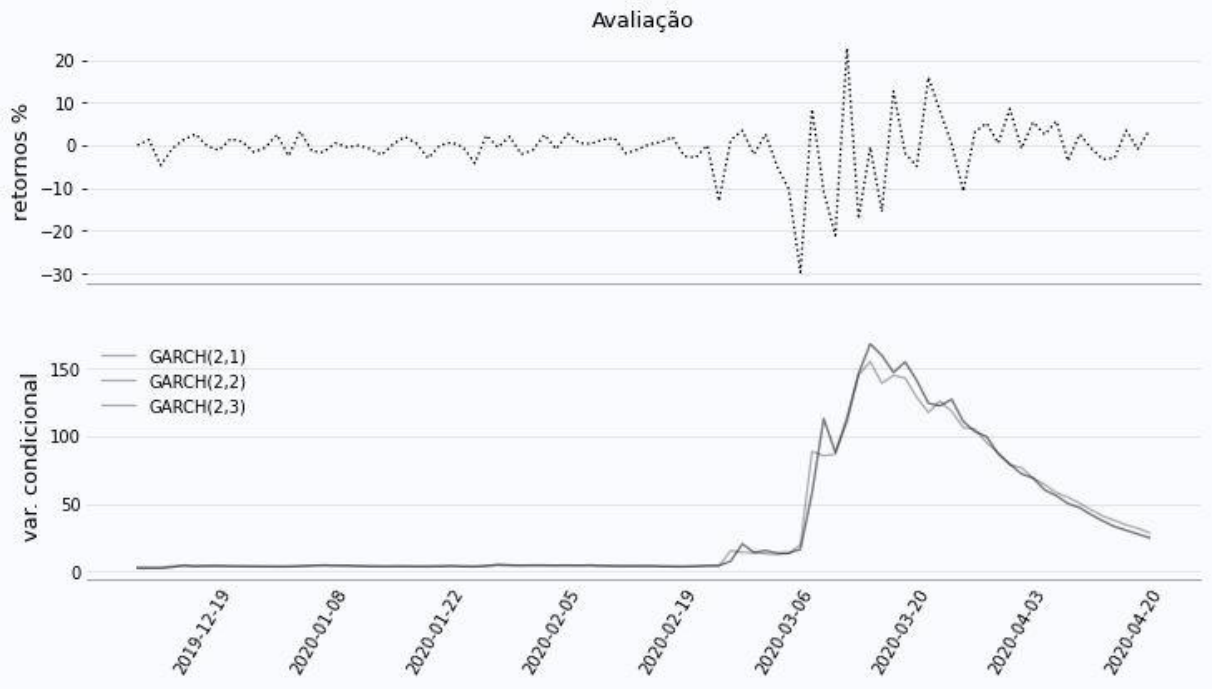
Gráfico 16: retornos modelo 3L-70N-LINEAR



Fonte: Elaboração própria

Os modelos sem o GARCH, em geral, preveem uma espécie de valor médio para o período em questão. Longe de ser um bom desempenho, o erro consegue ser menor que o *benchmark* nesses modelos. Com a introdução da previsão da variância fornecida pelos GARCHs, as previsões passam a estar completamente erradas, em alguns modelos é previsto retornos de mais de 1000%, sendo o valor máximo para período de aproximadamente 25%. Esses modelos conseguem, em algum nível, antecipar o aumento da volatilidade que acontece próxima ao dia 2020-03-06. Porém, foi dado um peso desproporcionalmente grande para esse evento, como resultado os retornos estão em uma escala de grandeza muito maior que a dos retornos.

Gráfico 17: retornos e variância condicional para o período de teste



Fonte: Elaboração própria

Em resumo, os melhores resultados foram alcançados pelas RNs sem GARCH. Seguido da média histórica como segundo melhor modelo preditivo e em terceiro temos o GARCH(2,3). As RNs com GARCH tiveram, de longe, o pior resultado.

2.4 - Considerações sobre a metodologia

Essa seção do trabalho é dedicada a comentários sobre as escolhas feitas durante a criação dos modelos, indicando o que pode ser feito para atingir melhores resultados em trabalhos futuros.

O uso de 3 sequências de variância condicional, uma de cada modelo GARCH, não parece ter sido uma boa idéia. Em geral, elas não são tão diferentes entre si, o que não fornece novas informações para a RN, gerando apenas ruído em seu *input*. Talvez uma possibilidade mais interessante fosse adicionar outros indicadores financeiros usados em ambiente de negociação como o volume diário, índice sharpe, etc.

O período de 113 observações para testar a capacidade preditiva foi uma escolha arbitrária e excessivamente longa. Uma abordagem mais interessante talvez fosse escolher uma janela de 21 (1 mês aproximadamente) ou 5 observações.

A RMSE como métrica de avaliação de *overfitting* também se mostrou problemática. Com um erro sempre menor para o conjunto de desenvolvimento em relação ao de treinamento, não foi possível avaliar se os modelos estavam sofrendo de *overfitting*, matando completamente o propósito da métrica nesse caso. Uma possível causa para esse fenômeno foi discutida na seção anterior. É necessário uma busca por outras métricas que consigam corrigir seu resultado pelo tamanho das sequências usadas como *input*.

O último ponto a ser destacado é o valor usado na taxa de aprendizado das RNs. Em todos os modelos foi usado o mesmo valor, o que não é uma boa prática (AURÉLIEN, 2019). Toda vez que um hiperparâmetro é alterado, como a quantidade de neurônios ou camadas, é necessário efetuar uma busca pela melhor taxa, o que não foi feito. O resultado disso é que os mínimos observados poderiam ser melhores, com modelos mais efetivos nas suas previsões e tempo de treinamento inferior.

CONCLUSÃO

Neste trabalho foi fornecido uma pequena introdução ao tema das RN artificiais, em específico das RN recorrentes com foco nas células tipo LSTM. Uma exposição similar foi feita para o modelo GARCH. Após essa introdução, uma abordagem que visava unir os 2 tipos de modelos foi testada.

Usando apenas os retornos para treinar as RNs, foram escolhidas as arquiteturas que apresentaram melhor performance preditiva. Com essas arquiteturas escolhidas, foi adicionado ao conjunto de treino a variância condicional fornecida pelos modelos GARCH e os modelos foram treinados novamente. Com isso foram avaliados os resultados dos novos modelos e, em geral, os resultados não foram muito promissores. Houve uma grande queda na performance preditiva de todos os modelos. Nos piores casos, os retornos da ação PETR3 previstos estavam completamente fora de escala, indicando retornos maiores que 1000% para um período que o retorno máximo não foi superior a 25%.

Dessa forma, os melhores resultados alcançados foram das RNs sem adição dos GARCH, o que indica que unir as 2 abordagens de forma produtiva não é tão simples. Podem existir muitas explicações para essa queda de desempenho, uma investigação que fugiria ao escopo do trabalho. Uma possível hipótese para essa queda é que as variâncias condicionais não fornecem um bom *input* para a previsão das RNs pois assumem apenas valores positivos, já os retornos podem assumir valores negativos ou positivos. Essa característica pode dificultar o mapeamento dos vetores de *input* para os valores corretos.

Existem muitas ideias a serem exploradas dentro desse paradigma da abordagem conjunta, como usar RNs para prever volatilidades e não os retornos. Porém, deve-se tomar cuidado para que matematicamente os *inputs* estejam em consonância com variáveis a serem previstas. Outros cuidados como a escolha da métrica correta também merecem melhor análise.

Bibliografia

- ARCH TOOLKIT: Documentação ARCH Toolkit para Python 3.x. Versão 4.15. [S.I.]: Kevin Sheppard, 2020. Disponível em: https://arch.readthedocs.io/. Acesso em: 25/06/2020.
- AURÉLIEN, Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2ª Edição. Sebastopol: O'Reilly, 2019.
- AWARTANI, Basel M. A.; CORRADI, Valentina. Predicting the volatility of the S&P-500 stock index via GARCH models: the role of asymmetries. International Journal of Forecasting. [S.I.], vol. 21, páginas 167-183, 2005.
- BOLLERSLEV, Tim; CHOU, Ray Y.; KRONER, Kenneth F. ARCH modeling in finance: A review of the theory and empirical evidence. Journal of Econometrics. North Holland, vol. 52, nº 1-2, páginas 5-59, abril-maio, 1992.
- BOLLERSLEV, Tim. Generalized autoregressive conditional heteroskedasticity. Journal of Econometrics. North Holland, vol. 31, nº 3, páginas 307-327, abril, 1986.
- CARR, Peter; WU, Liuren; ZHANG, Zhibai. Using Machine Learning to Predict Realized Variance. 2019. 15 páginas. NYU Tandon School of Engineering - New York.
- COWPERTWAIT, Paul S. P.; METCALFE, Andrew V. Introductory Time Series with R. 1ª Edição. New York: Springer, 2009.
- DONALDSON, R. Glen; KAMSTRA, Mark. An artificial neural network-GARCH model for international stock return volatility. Journal of Empirical Finance. [S.I.], vol. 4, páginas 17-46, 1997.
- DORFFNER, Georg. Neural Networks for Time Series Processing. Neural Network World. Vienna, vol. 6, páginas 447-468, 1996.
- ENGLE, Robert F.; BOLLERSLEV, Tim. Modelling the persistence of conditional variances. Econometric Reviews. [S.I.], vol. 5, nº 1, páginas 1-50, 1986.
- ENGLE, Robert F. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. Econometrica. [S.I.], vol. 50, nº 4, páginas 987-1007, julho, 1982.
- ENGLE, Robert F. GARCH 101: The Use of ARCH/GARCH Models in Applied Econometrics. Journal of Economic Perspectives. [S.I.], vol. 15, nº 4, páginas 157-168, 2001.
- FRANK, R. J.; DAVEY, N.; HUNT, S. P. Time Series Prediction and Neural Networks. Journal of Intelligent and Robotic Systems. Holanda, vol. 31, páginas 91-103, maio, 2001.
- HOCHREITER, Sepp; SCHMIDHUBER, Jurgen. Long short-term memory. Neural Computation. [S.I.], vol. 9, nº 8, páginas 1735-1780, 1997.
- HORNIK, Kurt; STINCHCOMBE, Maxwell; WHITE, Halbert. Multilayer

Feedforward Networks are Universal Approximators. Neural Networks. [S.I.], vol.2, páginas 359-366, 1989.

- HU, Michael Y.; TSOUKALAS, Christos. Combining conditional volatility forecasts using neural networks: an application to the EMS exchange rates. Journal of International Financial Markets, Institutions and Money. [S.I.], vol. 9, nº 4, páginas 407-422, novembro, 1999.
- KAASTRA, Iebeling; BOYD, Milton. Designing a neural network for forecasting and economic time series. Neurocomputing. [S.I.], vol. 10, páginas 215-236, 1996.
- KALMAN, Barry L.; KWASNY, Stan C. High performance training of feedforward and simple recurrent networks. Neurocomputing. [S.I.], vol. 14, páginas 63-83, 1997.
- LU, Xunfa; QUE, Danfeng; CAO, Guangxi. Volatility Forecast Based on the Hybrid Artificial Neural Network and GARCH-type Models. Procedia Computer Science. [S.I.], vol. 91, páginas 1044-1049, 2016.
- MANTRI, Jibendu Kumar; Gahan, P.; NAYAK, B. B. Artificial neural networks: an application to stock market volatility. International Journal of Engineering Science and Technology. [S.I.], vol. 2, nº 5, páginas 1451-1460, 2010.
- MCCULLOCH, Warren S.; PITTS, Walter. A Logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics. [S.I.], vol. 5, páginas 115-133, 1943.
- MONFAREDA, Soheil Almasi; ENKE, David. Volatility Forecasting using a Hybrid GJR-GARCH Neural Network Model. Procedia Computer Science. Philadelphia, vol. 36, nº 4, páginas 246-253, 2014.
- NELSON, Daniel B. Conditional Heteroskedasticity in Asset Returns: A New Approach. Econometrica. [S.I.], vol. 59, nº 9, páginas 347-370, março, 1991.
- PAGAN, Adrian R.; SCHWERT, G. William. Alternative models for conditional stock volatility. Journal of Econometrics. North-Holland, vol. 45, páginas 267-290, 1990.
- POON, Ser Huang; GRANGER, Clive W. J. . Forecasting Volatility in Financial Markets: A Review. Journal of Economic Literature. [S.I.], vol. 41, nº 2, páginas 478-539, junho, 2003.
- PORTUGAL, M. S.; FERNANDES, L. G. L. Redes neurais artificiais e previsão de séries econômicas: uma introdução. Nova Economia, [S. l.], vol. 6, nº 1, 2013.
- PORTUGAL, M. S. Neural Networks Versus Time Series Methods: A Forecasting Exercise. Revista Brasileira de Economia, vol. 49, nº 4, páginas 611-629, 1995.
- ROSENBLATT, Frank. Perceptron Simulation Experiments. Proceedings of the IRE. [S.I.], vol. 48, páginas 301-309, 1960.
- ROSENBLATT, Frank. The Perceptron: A Perceiving and Recognizing Automaton. Cornell Aeronautical Laboratory Inc Report. Buffalo, nº 85-460-1, páginas 1-29, janeiro, 1957.
- RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning internal representations by error propagation. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

- SAK, Hasim; SENIOR, Andrew; BEAUFAYS, Françoise. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. Google. [S.I.], páginas 1-5, fevereiro, 2014.
- SCHMIDHUBER, Jurgen. Deep learning in neural networks: An overview. Neural Networks. [S.I.], vol. 61, páginas 85-117, 2015.
- SMITH, Leslie N. A disciplined approach to neural network hyper-parameters: Part 1 - Learning rate, batch size, momentum, and weight decay. US Naval Research Laboratory Technical Report. Washington, nº 5510-026, páginas 1-21, abril, 2018.
- STATSMODELS: Documentação Statsmodels para Python 3.x. Versão 0.11.1. [S.I.]: Josef Perktold, Skipper Seabold, Jonathan Taylor, 2019. Disponível em: https://www.statsmodels.org/. Acesso em: 26/06/2020.
- TENSORFLOW: Documentação TensorFlow. Versão 2.x. [S.I.]: Google, 2020. Disponível em: https://www.tensorflow.org/. Acesso em: 25/06/2020.
- WHITE, Halbert. Economic prediction using neural networks: the case of IBM daily stock returns. In: INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, 1988, San Diego. [S.I.], IEEE, vol. 2, páginas 451-458, 1988.
- YIM, Juliana. A Comparison of Neural Networks with Time Series Models for Forecasting Returns on a Stock Market Index. In: INTERNATIONAL CONFERENCE ON INDUSTRIAL AND ENGINEERING, nº 15, 2002, Cairns. Developments in Applied Artificial Intelligence, Applications of Artificial Intelligence and Expert Systems. Local: Cairns, 2002. Páginas 25-35.
- ZAREMBA, Wojciech; SUTSKEVER, Bya; VINYALS, Oriol. Recurrent neural network regularization. In: International Conference on Learning Representations, nº 2, 2015, Vienna.

GLOSSÁRIO

TERMO	SIGNIFICADO
AIC	Critério de informação de Akaike
AR	Modelo autoregressivo
ARIMA	Modelo autoregressivo, integrado com médias móveis
Batch	Técnica de treinamento de redes neurais em que todos os exemplos são vistos para que a rede atualize uma vez seus parâmetros
Benchmark	Modelo usado como referência de performance
BIC	Critério de informação bayesiano
Dickey-Fuller aumentado	Teste para verificar a presença de raiz unitária na série
Epoch	Em redes neurais, representa quantas vezes todos os exemplos do conjunto de treino serão vistos pelo modelo
Função de ativação	Função usada ao final de todo neurônio de uma rede neural
Função de custo	Função que computa o erro da saída de uma rede neural
GARCH	Modelo generalizado autoregressivo de heterocedasticidade condicional
LSTM	Neurônio de memória longa e curta
MA	Modelo de médias móveis
Mini-batch	Técnica de treinamento de redes neurais em que apenas uma parte dos exemplos são vistos para que a rede atualize uma vez seus parâmetros
MLP	Rede neural composta de diversas camadas e neurônios
MSE	Média do erro ao quadrado

TERMO	SIGNIFICADO
Neurônio recorrente	Neurônio que possui memória
Overfitting	Quando o modelo não possui capacidade de generalização
Perceptron	Modelo inspirado no neurônio humano
PETR3	Série com os retornos da ação ordinária da Petrobrás S.A.
ReLU	Função de ativação linear para apenas valores positivos
RMSE	Raiz da média do erro ao quadrado
RN	Rede neural
RNR	Rede neural recorrente
Sigmóide	Função de ativação com formato em “S”
Softmax	Função de ativação usada em problemas de classificação com mais de duas classes
TANH	Função de ativação da tangente hiperbólica
Taxa de aprendizagem	Valor usado na descida de gradiente, define o tamanho do passo dado a cada iteração do algoritmo
Var(x)	Variância da série

ANEXO I - Saídas GARCH modelos selecionados

Saída GARCH(2,1)

Constant Mean - GARCH Model Results

Dep. Variable:	ROI	**R-squared:**	-0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	-0.000
Vol Model:	GARCH	**Log-Likelihood:**	-11512.8
Distribution:	Normal	**AIC:**	23035.6
Method:	Maximum Likelihood	**BIC:**	23068.3
		No. Observations:	5021
Date:	Tue, Aug 11 2020	**Df Residuals:**	5016
Time:	11:32:12	**Df Model:**	5

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	0.1157	3.376e-02	3.428	6.082e-04	[4.955e-02, 0.182]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.1274	3.448e-02	3.695	2.196e-04	[5.984e-02, 0.195]
alpha[1]	0.0658	9.838e-03	6.693	2.193e-11	[4.656e-02,8.512e-02]
beta[1]	0.9149	0.156	5.868	4.419e-09	[0.609, 1.221]
beta[2]	0.0000	0.154	0.000	1.000	[-0.302, 0.302]

Covariance estimator: robust

Saída GARCH(2,2)

Constant Mean - GARCH Model Results

Dep. Variable:	ROI	**R-squared:**	-0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	-0.000
Vol Model:	GARCH	**Log-Likelihood:**	-11505.7
Distribution:	Normal	**AIC:**	23023.4
Method:	Maximum Likelihood	**BIC:**	23062.5
		No. Observations:	5021
Date:	Tue, Aug 11 2020	**Df Residuals:**	5015
Time:	11:33:18	**Df Model:**	6

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	0.1148	3.347e-02	3.431	6.003e-04	[4.925e-02, 0.180]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.2128	6.448e-02	3.301	9.638e-04	[8.646e-02, 0.339]
alpha[1]	0.0231	1.392e-02	1.659	9.715e-02	[-4.192e-03,5.038e-02]
alpha[2]	0.0786	2.177e-02	3.610	3.059e-04	[3.593e-02, 0.121]
beta[1]	0.5600	0.153	3.659	2.533e-04	[0.260, 0.860]
beta[2]	0.3061	0.145	2.113	3.462e-02	[2.215e-02, 0.590]

Covariance estimator: robust

Saída GARCH(2,3)

Constant Mean - GARCH Model Results

Dep. Variable:	ROI	**R-squared:**	-0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	-0.000
Vol Model:	GARCH	**Log-Likelihood:**	-11505.7
Distribution:	Normal	**AIC:**	23025.4
Method:	Maximum Likelihood	**BIC:**	23071.0
		No. Observations:	5021
Date:	Tue, Aug 11 2020	**Df Residuals:**	5014
Time:	11:33:18	**Df Model:**	7

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	0.1148	3.448e-02	3.330	8.675e-04	[4.725e-02, 0.182]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.2128	0.876	0.243	0.808	[-1.505, 1.930]
alpha[1]	0.0231	3.322e-02	0.695	0.487	[-4.203e-02,8.821e-02]
alpha[2]	0.0786	2.269e-02	3.464	5.319e-04	[3.413e-02, 0.123]
alpha[3]	5.7128e-11	0.425	1.345e-10	1.000	[-0.832, 0.832]
beta[1]	0.5600	4.280	0.131	0.896	[-7.828, 8.948]
beta[2]	0.3061	3.750	8.164e-02	0.935	[-7.044, 7.656]

Covariance estimator: robust