Algoritmo Neuroevolutivo Aplicado na Predição dos Valores de Ações da Petrobrás

Aluno: Jherson Haryson Almeida Pereira
Matricula: 201611140004

Aluno: Danilo Souza Duarte
Matricula: 201611140020

Resumo

Predizer o comportamento das ações na bolsa de valores é uma tarefa desafiadora, muitas vezes relacionada a fatores desconhecidos ou influenciados por variáveis de naturezas bem distintas. No intuito de se antecipar a tais variações, já foram propostos mecanismos baseados em Redes Neurais Artificiais(RNAs) para predizer o comportamento de ativos na bolsa de valores, baseados em dados de indicadores pré-existentes. A precisão e acurácia de muitos desses sistemas baseados em RNAs dependem diretamente desses indicadores e de outros parâmetros utilizados em seu treinamento, que normalmente são ajustados por especialistas humanos através de um processo de tentativa e erro. Este processo é custoso e leva a incerteza quanto a melhor solução de um modelo de arquitetura escolhido.

Este trabalho busca desenvolver um algoritmo neuro evolutivo baseado em células de memória de longo prazo para aperfeiçoar um modelo de RNA recorrente, gerando topologias de RNAs automaticamente a fim de predizer dos valores das ações da Petrobras(PETR4) na bolsa de valores durante o ano de 2018.

Introdução

O projeto de uma Rede Neural Artificial (RNA) pode ser considerado um processo complexo de tomada de decisão que frequentemente depende da experiência do usuário. Em geral, essa tarefa de projeto de RNA é um processo de tentativa e erro, em que um número de diferentes funções de transferência e quantidade de neurônios devem ser ajustados para resolver um problema específico. Como o projeto de RNA, ainda é feito, manualmente, a automação deste processo de projeto beneficia o processo de tomada de decisão feito por especialistas humanos (Bukhtoyarov e Semenkin, 2013).

Metodologia

Foram realizados experimentos de predição dos valores de ações da Petrobras utilizando deep-learning com redes Long Short Term Memory(LSTM). A memória de longo prazo (LSTM) utilizada é uma arquitetura artificial de rede neural recorrente (RNN) utilizada no campo da aprendizagem profunda. Diferente das redes neurais de avanço padrão, o LSTM possui conexões de feedback (memórias). Não processando pontos de dados únicos, mas sim sequências inteiras de dados.

Uma das aplicações, do LSTM é em tarefas como reconhecimento de escrita manuscrito não segmentado, reconhecimento de fala e detecção de anomalias no tráfego de rede ou IDS (sistemas de detecção de intrusão).

A utilização de celulas LSTM é justificada justamente pelo fato que o histórico dos valores das ações influenciam no preço atual. Dessa forma o histórico deve ser levado em consideração.

Para a realização do projeto, foi utilizada a linguagem Python. Dessa forma, é necessário ressaltar que foram utilizadas algumas bibliotecas para facilitar a leitura de dados, implementação de RNAs, geração de gráficos, entre outros.

A primeira das bibliotecas utilizadas foi a biblioteca Pandas. Esta biblioteca tem como objetivo a análise, leitura e manipulação de grandes quantidades de dados. Ela foi utilizada especialmente para a leitura da base.

A segunda biblioteca utilizada foi o Scikit-learn, também conhecida como sklearn. Esta é uma biblioteca de código aberto voltada para técnicas de aprendizado de máquina. Esta biblioteca foi utilizada para a normalização dos dados utilizando a função MinMaxScaler, fazendo com que os dados fiquem entre 0 e 1 para que a convergência da rede seja melhor.

A terceira biblioteca utilizada foi o Keras. Esta é uma biblioteca que tem como foco a experimentação rápida de redes neurais e tende a ser fácil de usar, modular e extensível. O Keras foi utilizado para a realização da predição de séries temporais. Dessa forma, a implementação da rede neural totalmente funcional se deu em poucas linhas de código resultando em uma implementação do algoritmo Long Short Term Memory(LSTM).

A quarta biblioteca utilizada foi matplotlib. Esta biblioteca serve para a criação de gráficos, ou seja, com a visualização dos dados. Pelo fato de ser uma tarefa de séries temporais, é necessário a criação de gráficos para mostrar os valores desejados e preditos pela rede, fazendo com que seja fácil de visualizar a eficácia da topologia da rede neural.

Algoritmos que realizam muitos processamentos, como RNAs, especialmente as recorrentes, tendem a ser mais demorados. Visto isso, foi utilizada uma técnica de multiprocessamento para que a execução da rede neural artificial seja realizada de forma mais rápida. Esta técnica está implementada na biblioteca Keras e agiliza a execução das RNAs. O KerasGPU é uma implementação do keras que utiliza a GPU do computador para auxiliar os calculos e acelerar o processamento.

Arquitetura

Rede Neural Artificial

A arquitetura da Rede Neural artificial será fixa, as quais todas serão redes recorrentes com aprendizagem profunda(deep learning). Abaixo segue a arquitetura que a rede neural terá:

- 4 camadas, sendo duas ocultas
- Função de perda: erro média quadrático
- batch size: 150 iterações

Algoritmo Genético

Para cada indivíduo do Algoritmo Genético implementará uma topologia de Rede Neural Artificial diferente. Com isso, os genes terão as seguintes informações:

- Numero de neurônios da 1ª C - Faixa estimada (heuristica): [90 ~ 110]
- Numero de neurônios da 2ª C - Faixa estimada (heuristica): [40 ~ 60]
- Numero de neurônios da 3ª C - Faixa estimada (heuristica): [40 ~ 60]
- 1ª Probabilidade de Dropout - Faixa estimada (heuristica): [2.5% ~ 3.5%]
- 2ª Probabilidade de Dropout - Faixa estimada (heuristica): [2.5% ~ 3.5%]
- 3ª Probabilidade de Dropout - Faixa estimada (heuristica): [2.5% ~ 3.5%]
- Otimizador - Faixa estimada (heuristica): [0 ~ 3] - 0 = rmsprop, 1 = adam
- Numero de épocas - Faixa estimada (heuristica): [100 ~ 120] - 0 = rmsprop, 1 = adam

Baseado nesses valores, o Algoritmo Genético tentará encontrar o melhor conjunto de parâmetros para que a RNA consiga predizer com o menor erro possível.

Descrição da base de dados

O dataset das ações da Petrobrás(PETR4) foi extraidos do site Yahoo Finanças, os dados da Petrobras foram extraídos e catalogados de acordo com o valor de abertura do dia e o valor de fechamento.

Essa base de dados possui 1267 dados que representam o valor diário da ação PETR4 entre os anos de 2013 e 2018.

As simulações utilizaram somente os valores de fechamento e a predição foi executada utilizando 90 passos para tras e prevendo 3 passos à frente.

Desenvolvimento

In [ ]:

```
# Aplicação de algoritmo neuroevolutivo para predição do mercado de ações da petrobras util

# Aluno:
# Jherson Haryson Almeida Pereira e Danilo Souza Duarte

# Definição e Imports de bibliotecas iniciais

import random
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import math as mt

GENES =  '10'
TARGET = ['NIN1', 'NIN2', 'NIN3', 'PROPB', 'PROB2', 'PROP3', 'OPT', 'EPOCS']

base = pd.read_csv("dataset/petr4-treinamento.csv")
base = base.dropna()
base_treinamento = base.iloc[:, 1:2].values
normalizador = MinMaxScaler(feature_range=(0,1))
base_treinamento_normalizada = normalizador.fit_transform(base_treinamento)

base_teste = pd.read_csv('dataset/petr4-teste.csv')
preco_real_teste = base_teste.iloc[:, 1:2].values

base_completa = pd.concat((base['Open'], base_teste['Open']), axis=0)

entradas = base_completa[len(base_completa) - len(base_teste)-90:].values
entradas = entradas.reshape(-1,1)
entradas = normalizador.transform(entradas)

X_teste = []
for i in range(90, 112):
    X_teste.append(entradas[i-90:i, 0])

X_teste = np.array(X_teste)
X_teste = np.reshape(X_teste, (X_teste.shape[0], X_teste.shape[1], 1))

previsores = []
preco_real = []

for i in range(90, 1242):
    previsores.append(base_treinamento_normalizada[i-90:i, 0])
    preco_real.append(base_treinamento_normalizada[i, 0])

previsores, preco_real = np.array(previsores), np.array(preco_real)

previsores = np.reshape(previsores, (previsores.shape[0], previsores.shape[1], 1))


print('working...')


# Implementação da classe Individuo
```

```python
class Individual(object):
    def __init__(self, chromosome):
        self.chromosome = chromosome
        self.model = None
        if chromosome:
            self.fitness = self.calc_fitness()

    @classmethod
    def create_cromossomo(self):
        return [self.mutated_genes(i) for i in range(0, 8)]

    @classmethod
    def mutated_genes(self, index):
        # TARGET = ['NIN1', 'NIN2', 'NIN3', 'PROPB', 'PROB2', 'PROP3', 'OPT', 'EPOCS']
        random.seed()
        if index is 0:
            return 90 + int(random.uniform(0, 1)*20) #return 90 + int(random.uniform(0, 1)*
        elif index is 1 or index is 2:
            return 40 + int(random.uniform(0, 1)*20) #return 40 + int(random.uniform(0, 1)*
        elif index is 3 or index is 4 or index is 5:
            return random.uniform(2.5, 3.5)
        elif index is 6:
            return random.randint(0, 3)
        else:
            return random.randint(100, 120)

    def cross_and_mutate(self, par2):
        child_chromosome = []

        for gp1, gp2 in zip(self.chromosome, par2.chromosome):
            prob = random.random()

            if prob < 0.5:
                child_chromosome.append(gp1)

            else:
                child_chromosome.append(gp2)

        if random.random() > .35:
            mutation = random.randint(0, 7)
            child_chromosome[mutation] = self.mutated_genes(mutation)

        return Individual(child_chromosome)

    def getModel(self):
        return self.model

    def get_fitness(self):
        previsoes = self.rna().predict(X_teste)
        previsoes = normalizador.inverse_transform(previsoes)
        mean = mt.fabs(preco_real_teste.mean() - previsoes.mean())
        self.predict = previsoes
        self.score = mean
        tf.keras.backend.clear_session()
        return mean

    def plot(self, generation):

        f = plt.figure()
        ax = f.add_subplot(1, 1, 1)
        ax.plot(preco_real_teste, color='green', label='Preco real')
```

```
        ax.plot(self.predict, color='red', label='Preco predito')
        ax.legend()
        #f.plot(preco_real_teste, color='green', label='Preco real')
        #f.plot(previsoes, color='red', label='Preco predito')


        plt.savefig('result/g_'+str(generation)+'_i_'+str(self.score)+'.png', format='png',
                    transparent=True, bbox_inches=None, pad_inches=0.1)
        plt.close(f)
        f.clear(True)

        file = open('result/g_'+str(generation)+'_i_'+str(self.score)+'.txt', 'w')

        file.write('Intermediaria 1: ' + str(self.chromosome[0]) + '\n')
        file.write('Intermediaria 2: ' + str(self.chromosome[1]) + '\n')
        file.write('Intermediaria 3: ' + str(self.chromosome[2]) + '\n')
        file.write('Dropout 1: ' + str(self.chromosome[3]) + '\n')
        file.write('Dropout 2: ' + str(self.chromosome[4]) + '\n')
        file.write('Dropout 3: ' + str(self.chromosome[5]) + '\n')
        file.write('Otimizador: ' + str(self.chromosome[6]) + '\n')
        file.write('Épocas: ' + str(self.chromosome[7]) + '\n\n\n')

        file.write('predict\n\n\n'+str(self.predict)+'\n\n\n\n\nreal\n\n\n'+str(preco_real_
        file.close()


    def rna(self):
        # [NIN1, NIN2, NIN3, PROPB, PROB2, PROP3, OPT, EPOCS]
        opt_by_chromosome = 'rmsprop' if self.chromosome[6] >= 1.5 else 'adam'

        regressor = Sequential()
        regressor.add(LSTM(units=self.chromosome[0], return_sequences=True, input_shape=(pr
        regressor.add(Dropout(self.chromosome[3]))
        regressor.add(LSTM(units=self.chromosome[1], return_sequences=True))
        regressor.add(Dropout(self.chromosome[4]))
        regressor.add(LSTM(units=self.chromosome[2]))
        regressor.add(Dropout(self.chromosome[5]))
        regressor.add(Dense(units=1, activation='linear'))
        regressor.compile(optimizer=opt_by_chromosome, loss='mean_squared_error', metrics=[
        regressor.fit(previsores, preco_real, epochs=self.chromosome[7], batch_size=100, ve
        self.model = regressor
        return regressor

    def calc_fitness(self):
        self.score = self.get_fitness()
        return self.score



# Implementação do Algoritmo Genético


def GeneticAlgorithm(size=10):
    generation = 1
    interations = 0
    population, fitness, bestfit, mean, std, ngeneration, sup_edge, inf_edge = [], [], [],

    for i in range(size):
        cromo = Individual.create_cromossomo()
        population.append(Individual(cromo))
```

```
while (interations < 10):
    interations = interations+1
    print('Interation '+str(interations)+'...')

    x = 0
    ring_perc = []
    population = sorted(population, key=lambda x: x.score, reverse=False)
    for ind in population:
        ind.plot(interations)

    for cromo in population:
        fitness.append(cromo.score)
        ring_perc.append(x + cromo.score)
        x += cromo.score


    bestfit.append(population[0].score)
    mean.append(np.mean(fitness))
    std.append(np.std(fitness))
    sup_edge.append(mean[len(mean) - 1] + std[len(std) - 1])
    inf_edge.append(mean[len(mean) - 1] - std[len(std) - 1])

    if population[0].score <= 0.01:
        ngeneration.append(generation + 1)
        break

    new_g = []

    s = int((10 * size) / 100)
    new_g.extend(population[:s])

    s = int((90 * size) / 100)

    for i in range(s):
        # prob2 = random.randint(0, 100)/100
        prob = random.randrange(0, int(ring_perc[-1]*100))/100
        apt_sum = 0
        parent1 = None

        for ind in population:
            apt_sum += ind.score
            if apt_sum >= prob:
                parent1 = ind
                break

        prob = random.randrange(0, int(ring_perc[-1]*100))/100
        apt_sum = 0
        parent2 = None

        for ind in population:
            apt_sum += ind.score
            if apt_sum >= prob:
                parent2 = ind
                break

        child = parent1.cross_and_mutate(parent2)
        new_g.append(child)


    population = new_g
    ngeneration.append(generation)
```

```
        generation += 1

    return bestfit, ngeneration[len(ngeneration)-1]


# Simulações

def plotGraphConfidence(meanAgGer, supAgEdge, infAgEdge, maxAgGer, word=""):
    fig, ax = plt.subplots()
    fig.set_size_inches(18.5, 10.5)
    plt.plot(range(maxAgGer), meanAgGer, color='orange', label='Genetic Algorithm')
    plt.fill_between(range(maxAgGer), supAgEdge, infAgEdge, color='orange', alpha=0.3)
    plt.legend()
    plt.title ('Fitness X Generations -  %s' %word)
    plt.ylabel('Fitness')
    plt.xlabel('Generations')

    plt.show()

def executeWords(WORD=""):
    global TARGET
    TARGET = WORD
    meanAg, maxAgGer = [], 0
    auxNGenerationAg, auxBestFitnessAg = [], []

    for i in range(0, 1):

        bestFitnessAg, bestAgGeneration = GeneticAlgorithm(10)
        meanAg.append(bestFitnessAg)

        auxNGenerationAg.append(bestAgGeneration)
        auxBestFitnessAg.append(bestFitnessAg)

        if bestAgGeneration > maxAgGer:
            maxAgGer = bestAgGeneration

    for currentarray in meanAg:
        if len(currentarray) < maxAgGer:
            for i in range(maxAgGer - len(currentarray)):
                currentarray.append(currentarray[-1])

    meanAg = np.transpose(meanAg)


    stdAgGer ,meanAgGer ,supAgEdge ,infAgEdge = [], [], [], []

    for i in meanAg:
        stdAgGer.append(np.std(i))
        meanAgGer = np.append(meanAgGer, np.mean(i))

        supAgEdge.append(meanAgGer[-1] + stdAgGer[-1])
        infAgEdge.append(meanAgGer[-1] - stdAgGer[-1])

    if len(meanAgGer) < maxAgGer:
        for i in range(maxAgGer - len(meanAgGer)):
            meanAgGer = np.append(meanAgGer, meanAgGer[-1])
            supAgEdge.append(supAgEdge[-1])
            infAgEdge.append(infAgEdge[-1])

    plotGraphConfidence(meanAgGer, supAgEdge, infAgEdge, maxAgGer, word=WORD)
    return auxNGenerationAg, auxBestFitnessAg
```

```
# Intervalo de confiança


nGenerationAg, bestFitnessAg = [], []

for word in (0, 1):
    print(word)
    auxNGenerationAg, auxBestFitnessAg = executeWords(word)
    nGenerationAg.append(auxNGenerationAg)
    bestFitnessAg.append(auxBestFitnessAg)

def plotBarChartAcc(agAccuracy):
        fig, ax = plt.subplots()
        fig.set_size_inches(18.5, 10.5)
        plt.bar([0.25, 1.25, 2.25], agAccuracy, width=0.25, label='Genetic Algorithm', colo
        plt.xticks([0.25, 1.25, 2.25],('WORD A', 'WORD B', 'WORD C'))
        plt.legend()
        plt.ylabel('hits')
        plt.title('hit percent per words')
        plt.show()

agAccuracy = []

for index in range(0, 1):
    agHit = 0

    for i in range(10):
        if bestFitnessAg[index][i][0] <= 0.02:
            agHit += 1

    agAccuracy.append(agHit*10)



plotBarChartAcc(agAccuracy)


# Geração por Palavra

# print(nGenerationAg)
# print(nGenerationRw)
# print(bestFitnessAg)
# print(bestFitnessRw)


def plotBarChart(meanAg, stdAg, ):
        fig, ax = plt.subplots()
        fig.set_size_inches(18.5, 10.5)
        plt.bar([0.25, 1.25, 2.25], meanAg,yerr = stdAg, width=0.25, label='Genetic Algorit
        plt.xticks([0.25, 1.25, 2.25],('WORD A', 'WORD B', 'WORD C'))
        plt.legend()
        plt.ylabel('Mean of generations')
        plt.title('Mean of generations per words')
        plt.show()

meanAg = [np.mean(nGenerationAg[i]) for i in range(3)]
stdAg = [np.std(nGenerationAg[i]) for i in range(3)]
```

```
plotBarChart(meanAg, stdAg)

print('| ACC   |    '+str(round(agAccuracy[0]))+'%  / ')
```

Resultados obtidos

O padrão fixado para a execução do Algoritmo Genético foram 10 gerações com 10 indivíduos cada, totalizando 100 simulações com redes LSTM em cada execução. Pelo fato de se tratar de uma tarefa de predição, não há como totalizar acertos e erros. Em virtude disso, a função fitness se deu a partir do cálculo do erro médio quadrático de cada RNA. O desempenho pode ser analisado no gráfico abaixo:

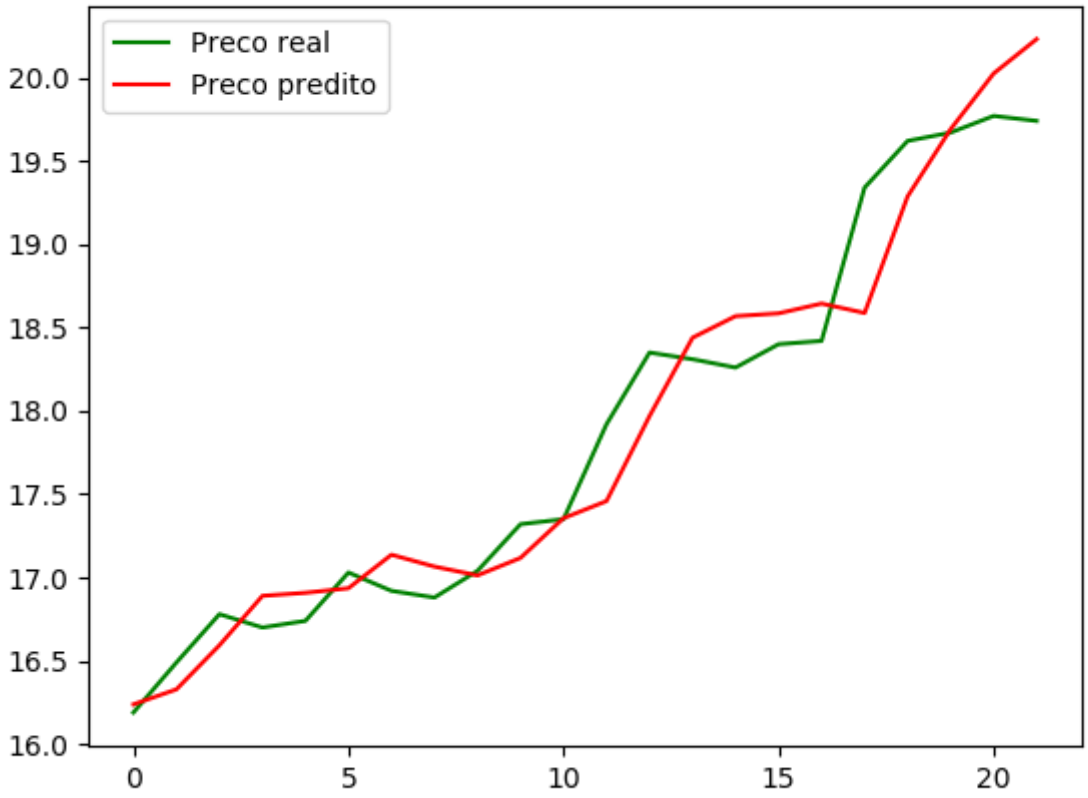

Figura 1 - Gráfico obtido do indivíduo 1, geração 1

Esta Rede Neural obteve um erro médio quadrático de 0.0077 e foi considerada a melhor de sua geração. No entanto, nota-se que o gráfico não acompanha o aumento ou decréscimo das ações, o que acaba por não agregar valor as informações, visto que é de suma importancia ao uauário saber a previsão de alta e de baixa das ações, indicando-o quando deve-se comprar ou vender as mesmas.
A topologia gerada para esta RNA foi a seguinte:

- Intermediaria 1: 101
- Intermediaria 2: 58
- Intermediaria 3: 58
- Dropout 1: 2.8681998005112854
- Dropout 2: 2.638358466953944
- Dropout 3: 2.6013069907521205
- Otimizador: rmsprop

- Épocas: 107

Por outro lado, o indivíduo 10 da geração 1 obteve o erro médio quadrático igual a 1.1274 mas seu gráfico acompanha exatamente as oscilações que o preço das ações obteve.

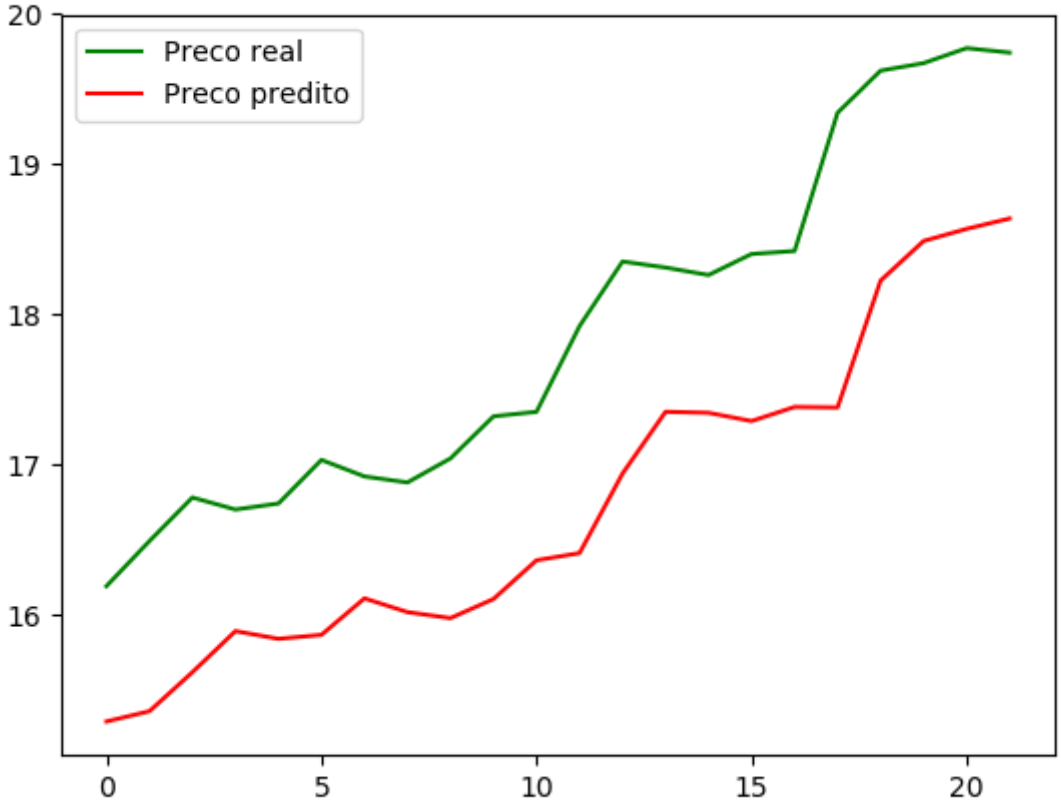

Figura 2 - Gráfico obtido do indivíduo 10, geração 1

Esta Rede Neural pertence a mesma geração da RNA da figura 1. No entanto, está foi a considerada pior. Nota-se que esta, apesar de não estar predizendo os preços de forma exata, acompanha a valorização e desvalorização dos preços das ações, porém, de acordo com o fitness calculado, não foi considerada a melhor. A topologia desta rede foi a seguinte:

- Intermediária 1: 107
- Intermediária 2: 50
- Intermediária 3: 42
- Dropout 1: 3.09869512109663
- Dropout 2: 2.5053067406952394
- Dropout 3: 3.4244695170709054
- Otimizador: adam
- Épocas: 117

As rede deep LSTM possuem uma facilidade natural para a predição de series temporais, já que conseguem relembrar dos dados anterios dessa forma chegou-se o seguinte resultado para o melhor indivíduo (4) na ultima geração(10).

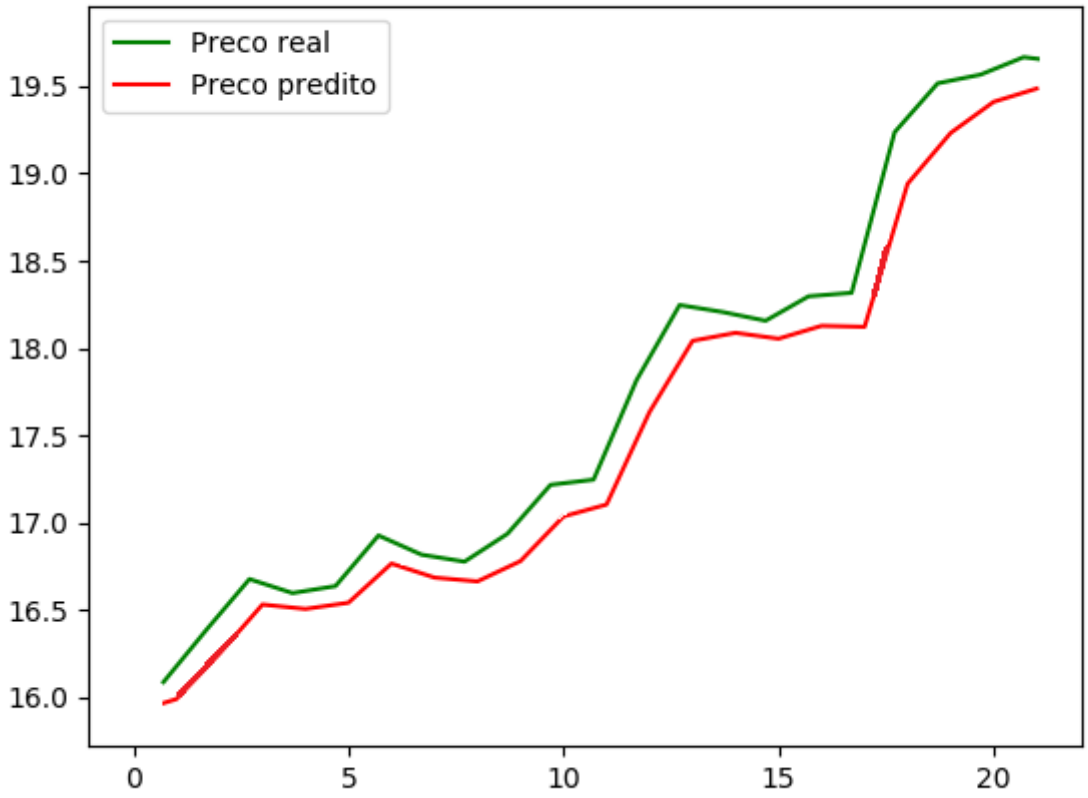

Figura 3 - Gráfico obtido do indivíduo 4, geração 10

A topologia da rede neural gerada foi a seguinte:

- Intermediária 1: 99
- Intermediária 2: 53
- Intermediária 3: 42
- Dropout 1: 3.09844212109663
- Dropout 2: 2.6073517406952394
- Dropout 3: 3.3244695170709054
- Otimizador: adam
- Épocas: 113

O indivíduo 4 da geração 10 obteve o erro médio quadrático igual a 0.0024 e seu gráfico acompanha exatamente as oscilações que o preço das ações obteve, ficando bastante aproximado do valor real.

Conclusão

O modelo de neuroevolução proposto conseguiu atingir as expectativas e prever os 3 passos a frente de maneira corretamente, ficando com valores bem aproximados aos valores reais. O sistema utilizou-se de redes LSTM para a previsão das ações, e um Algoritmo genetico para evoluir design de arquiteturas de RNA automáticamente(ADEANN). O custo de processamento é bastante elevado, para isso utilizou-se a bliblioteca Keras-GPU para processamento através de placas gráficas. Observou-se que o tempo de execução foi reduzido em 20% como pode ser visto no grafico abaixo, mostrando o eixo y em milisegundos.

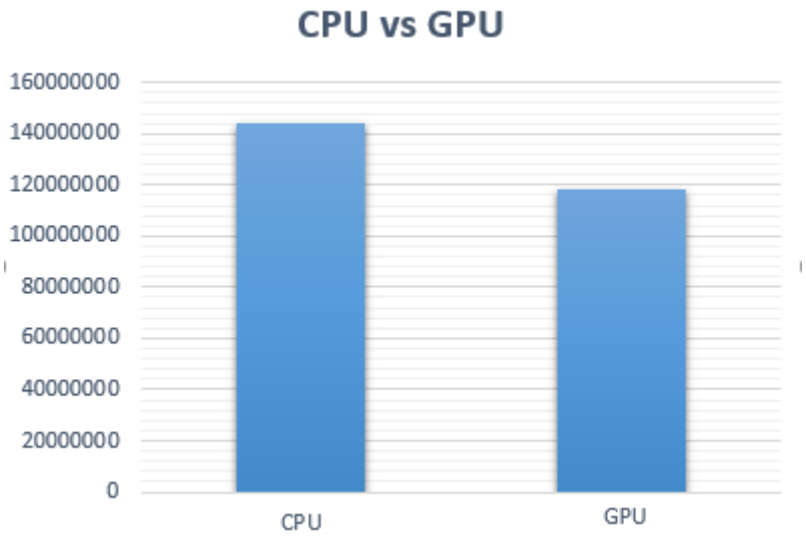

Figura 4 - Gráfico de desempenho CPU vs GPU (Keras vs KerasGPU)

Uma simulação utilizando a CPU demorou aproximadamente 40 horas para ser finalizando, utilizando 10 gerações com 10 indivíduos, em contra partida utilizando a ajuda da GPU, para as mesmas configurações do exemplo anterior, a simulação executou com uma estimatica de 32 horas, mostrando ser bem menos custosa.

Trabalhos Futuros

O custo do algoritmo desenvolvido é naturalmente elevado devido a sua codificação direta da arquitetura de rede, para melhorar a performance pode ser implementado um modelo de multiprocessamento e processamento paralelo para otimizar o tempo de execução, visto alguns sistemas nescessitam de respostas mais rápidas não sendo possivel ter "tempo" para o treinamento e seleção do modelo. Outro ponto também que também pode auxiliar no desenvolvimento do modelo é a utilização de uma codificação indireta de RNA através de uma gramática, dessa forma o AG pode evoluir uma gramática que representa um modelo de rede neural. Esse tipo de abordagem aumenta o espaço de busca e diminui o custo de exploração.