Inteligência Computacional

Trabalho de Implementação de Algoritmo Genético com Mistura Hieráquica de Especialistas

Michel Monteiro Zerbinati - Nº USP: 6145570

Introdução

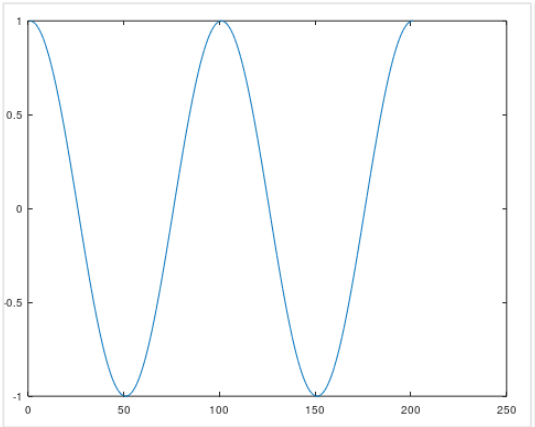
Este trabalho abordou a utilização do Algoritmo Genético para a identificação de uma boa arquitetura de Mistura Hierárquica de Especialistas, considerando 2 séries que foram utilizadas nos experimentos abaixo.

O foco principal que dei foi na geração e criação das possíveis arquiteturas que um MHE poderia ter e, que ao longo da execução do Algoritmo Genético, as arquiteturas de MHE obtivessem melhor Fitness, que neste trabalho utilizei a maximização do likelihood (lli), pudessem se manter e propagar seus genes com maior probabilidade para as seguintes gerações.

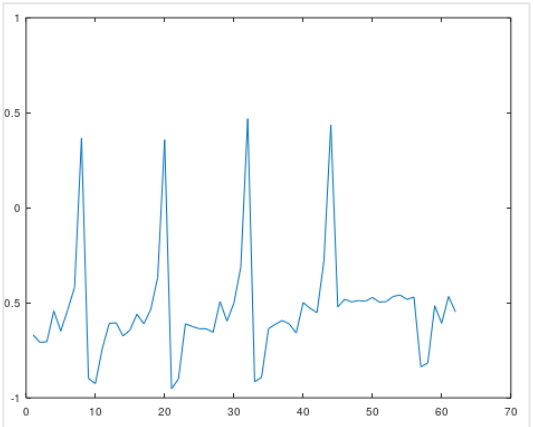
Este documento está estruturado da seguinte forma:

- Introdução
- Séries utilizadas (gráficos das séries)
- Estratégias utilizadas (utilzei 3 formas de variação da arquitetura da MHE)
- Detalhamento e Resultado das estratégias utilizadas
- Características técnicas das execuções
- Conclusão

Séries utilizadas



Dados2



Serie1

Estratégias utilizadas

Para este trabalho foi desenvolvido um algoritmo genético em Python e implementadas algumas codificações de cromossomos para que fosse consumido o código de Mistura Hierárquica de Especialistas disponibilizado em MatLab/Octave.

Inicialmente, estruturei este trabalho em 3 principais experimentos e estratégias de geração e evolução da arquitetura da MHE:

- Variação somente da Profundidade e Ramificação da arquitetura da MHE
- Variação da Profundidade, da Ramificação dos Gatings da MHE, e da Ramificação dos Especialistas (folhas) de cada Gating, de forma independente
- Variação total da arquitetura da MHE

Detalhamento e Resultado das Estratégias

A. Variação somente da Profundidade e Ramificação da MHE

Neste experimento procurei codificar um cromossomo de tal forma que pudesse representar somente a Profundidade e a Ramificação da estrutura da árvore da MHE. Sendo assim, o cromossomo ficou com 4 bits (0 0 0 0), sendo os 2 primeiros para altura, e os 2 últimos bits para o fator de ramificação.

Para garantir que fossem apresentados dados sempre factiveis, ou seja, altura > 0 e fator de ramificação >= 2 (pois um gating, somente com 1 filho, poderia ser considerado um especialista), apliquei a seguinte regra para decodificação do binário para o número inteiro:

- Produnfidade => bit_decodificado **+ 1**
  - desta forma, os valores 0 0 (0), 0 1 (1), 1 0 (2) e 1 1 (3) se tornam **1, 2, 3 e 4**
- Ramificação => bit_decodificado **+ 2**
  - Os valores 0 0 (0), 0 1 (1), 1 0 (2) e 1 1 (3) se tornam **2, 3, 4 e 5**

Assim, temos ums espaço de busca entre (1,2) e (4,5). Apesar de a quantidade de possibilidades não serem um espaço muito grande, limitei a esses valores devido a demora na execução ao aumentarmos essas dimensões.

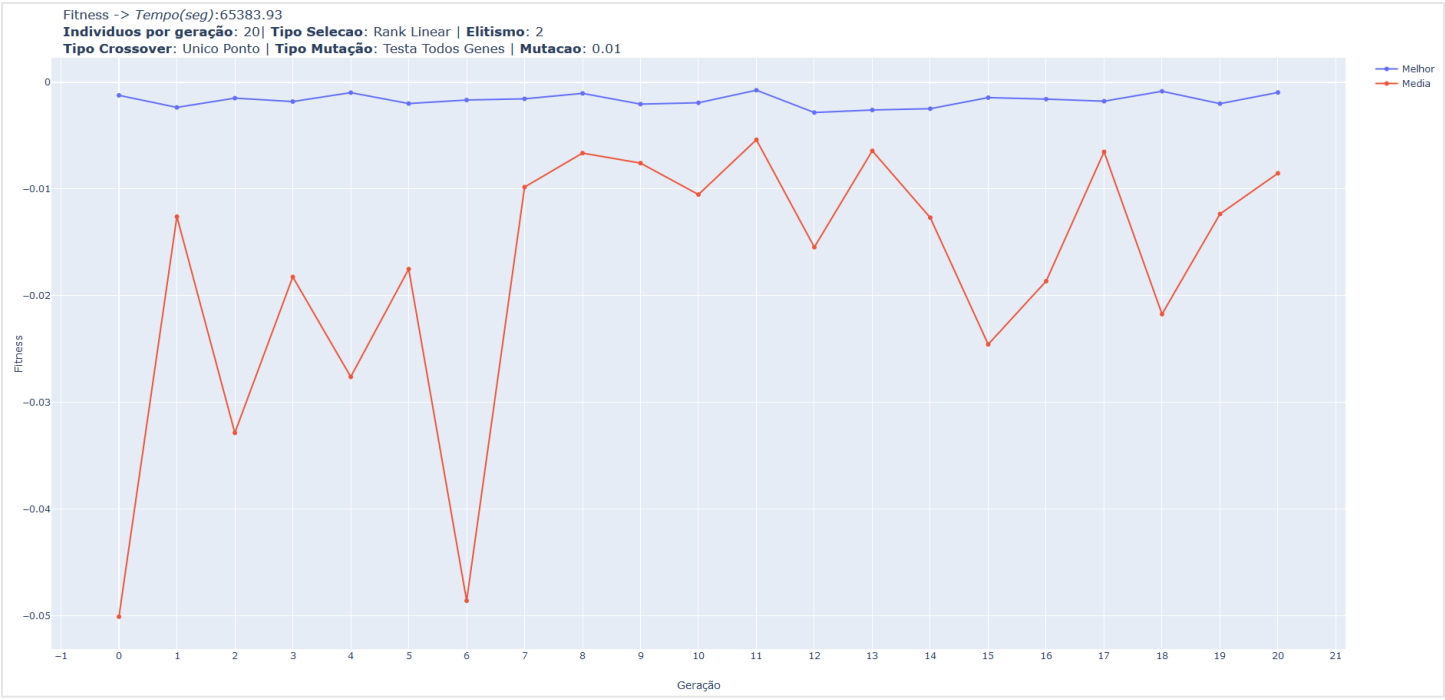
Para que esta codificação pudesse ser utilizada e integrada entre os códigos em Python e MatLab/Octave, a seguinte alteração no código htmTest.m foi realizada:

```
function [hme_fit, yexp, z]= hmeTest(profundidade, ramificacao)
    depth=profundidade; % profundidade
    factor=ramificacao; % fator de ramificação
    nexperts=factor; % fator de ramificação
```

Com isso, após a decodificação dos cromossomos, o código MatLab era chamado recebendo os valores da profundidade e ramificação da árvores, para que o Fitness fosse calculado.

Abaixo, alguns experimentos realizados e seus respectivos gráficos:

***Gerações:** 20 | **Individuos por Geração:** 20 | **Tipo Seleção:** Rank Linear | **Crossover**: Único Ponto | **Mutação:** Verifica Todos os Genes | **Mutação**: 0.01*

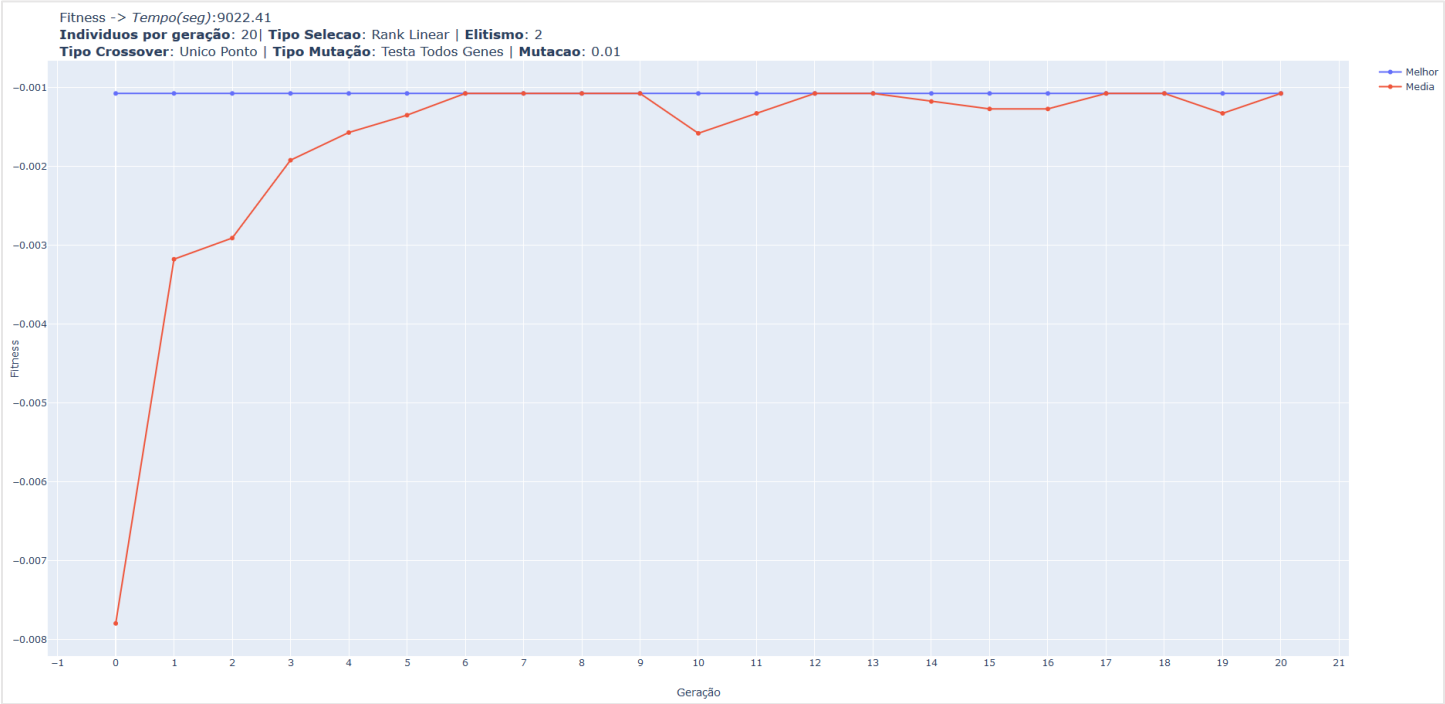


Dados2

Neste gráfico podemos visualizar que, mesmo utilizando o Elitismo, o melhor indivíduo não se mantem com o valor de Fitness constante. Isso se dá devido a “aleatoridade” da geração dos pesos iniciais para os especialistas e gatings da MHE, fazendo com que, a cada execução, ela gere uma valor diferente. Para esta execução, a estrutura de MHE que obteve o melhor resultado foi a (1,3), ou seja, profundidade 1 e ramificação 3, que reflete a uma rede gating como nó e com 3 especialistas. Essa arquitetura alcançou o melhor Fitness de todas as gerações na geração 11 (Fitness: -0.000769) e, na última, a geração 20, também foi o melhor indivíduo com o Fitness -0.000981.

Como forma de experimento, também implementei uma maneira de preservar os valores das execuções das estruturas que já foram executadas para reaproveitá-las depois, para ver o comportamento do gráfico gerado e a melhora que poderia oferecer no tempo de processamento. Por exemplo, caso a estrutura (2,3) tenha sido executada na geração 1 e tenha gerado o valor de Fitness -0.20, este valor foi armazenado e, nas próximas gerações, caso esta estrutura (2,3) seja gerada como um novo cromossomo, ao invés de reexecutá-la novamente esta MHE, capturo este valor (-0.20) e já atribuo para este indivíduo como Fitness. Foi possível perceber que, para este cenário, o tempo de execução diminuiu consideravelmente (de +/- 18hrs para +/- 2h30m), e gerou o seguinte gráfico abaixo:

***Gerações:** 20 **| Individuos por Geração:** 20 **| Tipo Seleção:** Rank Linear **| Crossover:** Único Ponto **| Mutação:** Verifica Todos os Genes **| Mutação:** 0.01*

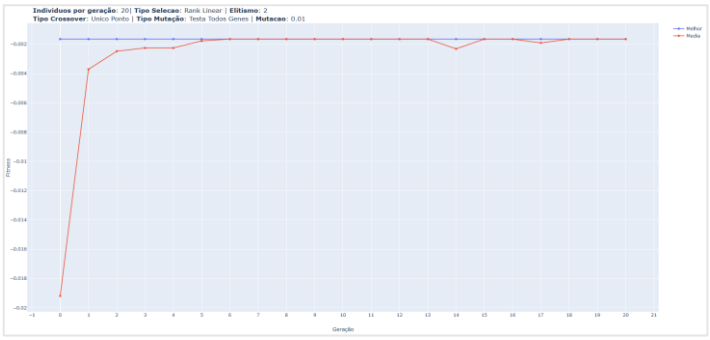


Dados2

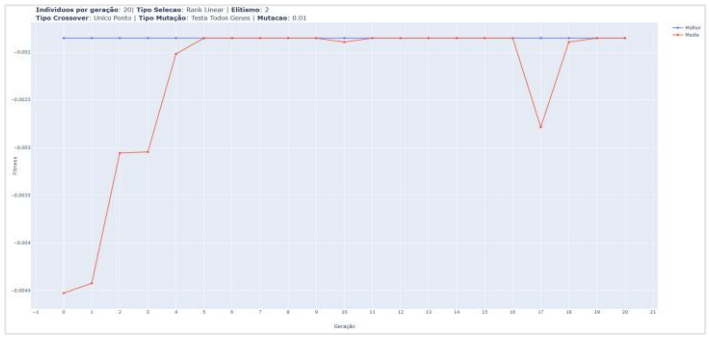
Podemos perceber que, o melhor indivíduo foi gerado já na 1º geração, alcançando o melhor valor de -0.0010, e com a estrutura (1,4), ou sejam, Profundidade 1 e Ramificação 4. Como estamos utilizando o Elitismos juntamente com o armazenamento dos resultados para cada estrutura, o melhor indivíduo que foi gerado na 1º geração não foi superado por outros indivíduos nas demais gerações, o que causou a constante do melhor indivíduo no gráfico acima, desde a geração 1 até a 20. Podemos perceber também que, entre as gerações 6 e 9, 12 e 13, e 17 e 18, a diversidade foi minima, o que causou, devido ao elitismo e também a forma como aplicamos o algoritmo (guardar os valores para cada arquitetura durante a execução), uma convergência para esta determinada arquitetura (1,4).

Para esta primeira variação, fiz mas 2 experimentos usando o armazenamento dos valores das arquiteturas já processadas, mas utilizando o espaço de busca entre (1,4) e (3,4):

Gerações: 20 | Individuos por Geração: 20 | Tipo Seleção: Rank Linear | Crossover: Único Ponto | Mutação: Verifica Todos os Genes | Mutação: 0.01



A (dados2)



B (dados2)

No experimento **A** a melhor arquitetura foi a (1,2) com o Fitness de -0.0016. No **B**, a melhor arquitetura também foi a (1,2), com o valor de Fitness = -0.0018. Assim, em ambas as melhores arquiteturas foram MHE de 1 Gating com 2 Especialistas.

B. Variação da Profundidade, Ramificação Gatings e Especialistas (folhas)

Neste experimento procurei codificar um cromossomo de tal forma que pudesse representar a Profundidade e Ramificação da estrutura de árvore da MHE (assim como no experimento A) mas também podendo variar a quantidade de especialistas abaixo dos últimos gatings da árvore, ou seja, variar o número de folhas da árvore. Assim, a estrutura inicial dos bits continua a mesma (para a Profundidade e para a Ramificação) e foram adicionados mais bits para representar a quantidade de especialistas abaixo de cada gating final, ou seja, as folhas. Devido ao aumento da quantidade de especialistas/folhas (limitei entre 2 e 5), diminui um nível de ramificação, ficando entre 2 e 3. Neste cenário, os bits da profundidade e ramificação ficaram (0 0 0), sendo os 2 primeiros sendo a profundidade e o último representando a ramificação. O espaço de busca desta árvore (ainda não considerando os especialistas folhas), será de (1,2) até (4,3), e utiliza a mesma regra de decodificação apresentada na estratégia anterior (Item A).

Para a quantidade total de especialistas de cada Gating final, utilizei 2 bits, gerando o espaço entre 2 e 5 especialistas para cada Gating ao final da árvore. Como podemos ver, o número de especialistas não foi limitado ao fator de ramificaçãode Gatings da árvore. Esse fator de ramificação da árvore será utilizado para serem criadas as estruturas dos gatings, e não das folhas.

Também, para evitar o risco de introduzir problema 'bloat', onde os filhos, após os crossovers, vão crescendo de tamanho, foi definida uma alocação de máximo de bits que seriam possíveis, utilizando a seguinte fórmula:

$$qtd\ bits\ especialistas = ramificacao\_max^{profundidade\_max-1} * 2$$

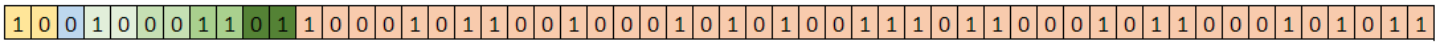
ou seja

$$qtd\ bits\ especialistas = 3^{4-1} * 2 = 54$$

Multiplicamos por 2 pois desejamos utilizar 2 bits para representar cada codifição da quantidade de especialistas para cada gating.

Ou seja, caso a estrutura da árvore codificada em um determinado indivíduo seja < 54, somente os bits "válidos" para aquela representação serão utilizados, e os demais, o qual considero "genes adormecidos" para esse indivíduo, não impactarão na sua decodificação. Eu os mantenho pois, em futuros crossovers, eles serão úteis e adicionarão diversidade às próximas gerações.

Abaixo um exemplo visual de uma codificação:



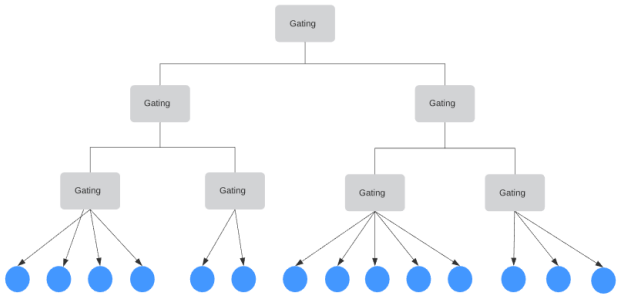
Altura = (1 0) = 2 => 2 + **1** = 3 (o 1 é o ajuste, igual utilizado no experimento A e explicado novamente abaixo)

Ramificação (0) = 0 => 0 + **2** = 2 (o 2 é o ajuste, igual utilizado no experimento A e explicado novamente abaixo)

Especialistas => Como no exemplo temos uma arvore (3,2), podemos aplicar a mesma fomula para saber qual é a quantidade de Gatings que precisamos definir a quantidade de especialistas, ou seja, 2^(3-1) => 2^2 => 4. Como sabemos que são 4 gatings que necessitam da quantidade de especialistas sejam definidos, decodificamos os próximos 8 bits do cromossomo, que acima estão em tons de verde. Assim, decodificando-os:

(1 0) = 2 => 2 + 2 = **4** | (0 0) = 0 => 0 + 2 = **2** | (1 1) = 3 => 3 + 2 = **5** | (0 1) = 1 => 1 + 2 = **3**

Assim teremos, respectivamente, 4, 2, 5 e 3 especialistas para cada Gating, conforme árvore abaixo:



Novamente, para garantir que fossem apresentados dados sempre factíveis, ou seja, altura >= 1 e fator de ramificação e de especialistas >= 2, apliquei a seguinte regra para decodificação do binário para o número inteiro:

- Altura => bits_decodificados **+ 1**
  - desta forma, os valores 0 0 (0), 0 1 (1), 1 0 (2) e 1 1 (3) se tornam **1, 2, 3 e 4**
- Ramificação => bits_decodificados **+ 2**
  - Os valores 0 (0), 1 (1) se tornam **2 e 3**
- Especialistas => bits_decodificados **+ 2**
  - Os valores 0 0 (0), 0 1 (1), 1 0 (2) e 1 1 (3) se tornam **2, 3, 4 e 5**

Para que essa codificação funcionasse no código do MatLab/Octave, efetuei as seguintes alterações:

htmTest.m

```
function [hme_fit, yexp, z]= hmeTest(profundidade, ramificacao, folhas)
    depth=profundidade; % Profundidade
    factor=ramificacao; % fator de ramificação
    nexperts=factor; % fator de ramificação

    % disp(sprintf('Carregando os dados'));
    load dados2
    global index_gating;
    index_gating = 1;

    [N,dim]=size(X); %J com bias
    [N,k]=size(Y);

    hme = hmeCreate(depth,nexperts,dim,k, folhas); %cria a estrutura
    hme = hmeInitRand(hme,depth,nexperts,dim,k);%Inicializa a estrutura
```

hmeCreate.m

```
global index_gating

if levels == 0,
    hme = hmeCreateExpert(d,k); % cria os especialistas
else
```

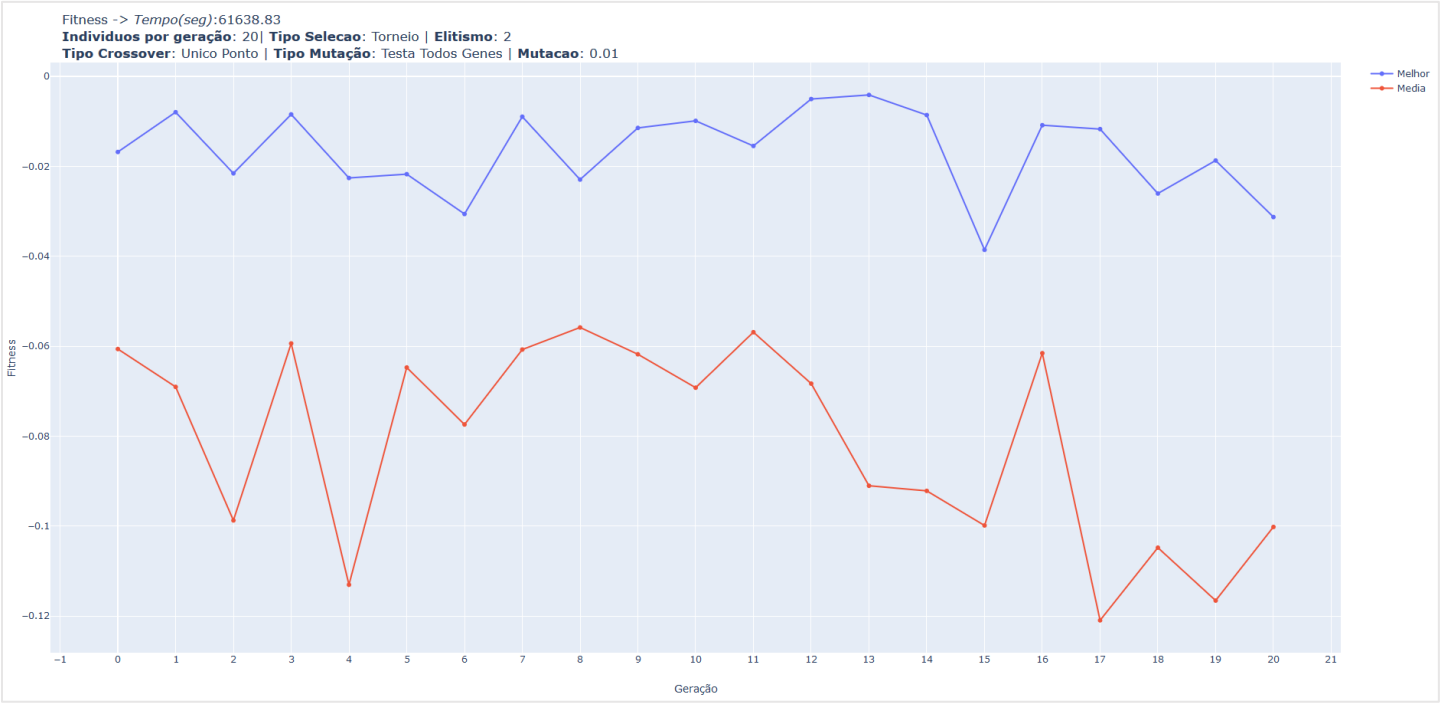
```
  if (levels-1)==0
    % EX.: em uma estrutura 2x3, uma arvore completa terá 9 especialistas(FOLHAS) ou seja [3
, 3 , 3]
    % O Vetor informará quantas folhas cada gating do nivel final terá:
    % Ex.: [2,3,2] informará que o Gating 1 tera 2 especialistas, o Gating 2 terá 3 e o
Gating 3 terá 2
    % Quantidade de EXPERTS
    ind_fol = index_gating;
    index_gating = ind_fol + 1;
    b = folhas(ind_fol);
    children = cell(1,b); %cria os filhos
  else
    children = cell(1,b); %cria os filhos
  end

  for i = 1:b,
    if (levels-1)==0
        % disp(sprintf('\n****************************************')) % MICHEL
        % disp(sprintf('Rede Especialista do ramo %d',i))       % MICHEL
        % disp(sprintf('****************************************')) % MICHEL
    end
    children{i} = hmeCreate(levels-1,b,d,k, folhas); % MICHEL_FOLHAS

  end
  % disp(sprintf('\n****************************************'))% MICHEL
  % disp(sprintf('Rede Gating do nivel %d',levels))       % MICHEL
  % disp(sprintf('****************************************')) % MICHEL
  hme = hmeCreateMixture(children,d);
end
```

hmeInitRand.m

```
else % Rede Gating
    [N,b] = size(hme.children); % MICHEL_FOLHAS - PEGA a QTDE de FILHOS. adicão dinamica das
folhas
    for i = 1:b, % MICHEL - ALTERAR AQUI para FUNCIONAR com folhas dinamicas
        hme.children{i} = hmeInitRand(hme.children{i},levels-1,b,d,k);
    end
```

Na próxima página, alguns experimentos realizados e seus respectivos gráficos:

Gerações: 20 | Individuos por Geração: 20 | Tipo Seleção: Torneio | Crossover: Único Ponto | Mutação: Verifica Todos os Genes | Mutação: 0.01



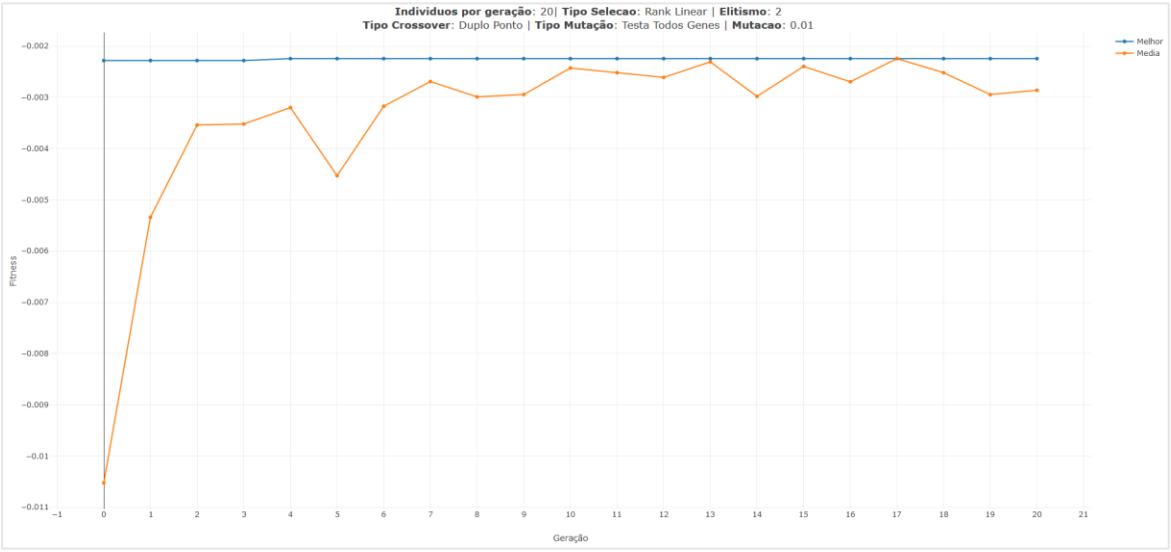
Dados2

No experimento acima, é possível visualizar que, ao decorrer das gerações, não foi alcançada uma convergência, e o gráfico destoa de outros gráficos comuns ao Algoritmo Genético. Isto ocorre pois, apesar de os melhors individuos da geração serem selecionados previamente para a próxima geração (elitismo = 2) e de que os demais melhores indivíduos terem maiores probabilidades de serem escolhidos e terem seus genes propagados, todos são reexecutados novamente na nova geração e, como nas Redes Neurais Especialistas e Gatings contidos dentro da MHE tem os pesos iniciados aleatoriamente, não existe garantia de que o mesmo resultado (seja o likelihood ou o EQM) se comportem da mesma maneira. Por este motivo, visualizamos o comportamento do gráfico acima, onde o melhor indivíduo oscila entre alta/baixa do fitness. Isso ocorre o mesmo com a média.

Ao final, a melhor arquitetura foi a de Profundidade = 1 com 2 Especialistas (1,2), com o Fitness de -0.004132180337512459, alcançado na 13º geração. Esta mesma arquitetura foi o melhor individuo nas gerações 7, 12, 14, 16, 17 e 19, nos comprovando que, apesar da variação de fitness entre as execuções, este é um bom padrão de arquitetura para os dados propostos.

Da mesma forma que o experimento anterior, nesta outra execução o valor de Fitness foi preservado para as arquiteturas entre as gerações, onde podemos notar que só houve uma mudança de posição (evolução) entre as gerações 3 e 4:

***Gerações:** 20 | **Individuos por Geração:** 20 | **Tipo Seleção:** Rank Linear | **Crossover**: Único Ponto | **Mutação:** Verifica Todos os Genes | **Mutação:** 0.01*



Dados2

Para esta execução, a melhor arquitetura foi a (3,2), com a respectiva quantidade de especialistas para os 4 gatings finais [3 5 5 3], e com fitness de -0.00224.

C. Variação Total da Estrutura da Rede

Com o intuito de efetuar testes com arquiteturas bem dinâmicas da MHE, os códigos do algoritmo genético e da MHE foram adaptados para que fosse possível representar diversos formatos de arquiteturas, podendo estas serem totalmente não balanceadas.

Foi necessário adaptar uma nova codificação, via matriz, para que pudessemos representar todos níveis da MHE, de acordo com um limite pré-definido de profundidade e ramificação máximos permitidos.

Sendo assim, a codificação do cromossomo utilizado pelo Algoritmo Genético ficou da seguinte forma, explicado a seguir:

1. Inicialmente, são definidios os limites máximos esperados da profundidade e ramificação que a MHE pode ter. Ex.: Profundidade = 2, Ramificação = 2
2. Com estas informações, a estrutura da matriz é criada, considerando o máximo tamanho que a árvore da MHE pode chegar. Para isso, utilizamos a seguinte fórmula:

$$qtd\_bits\_profundidade = \sum_{i=0}^{profund_{max}-1} qtd\_bits\_profundidade + (ramificacao\_max^{i} * ramificacao\_max)$$

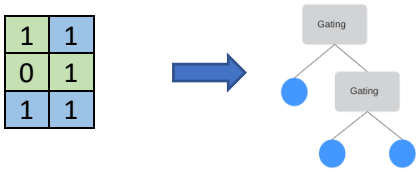
3. Com isto, é possível definir o número de linhas da matriz do cromossomo. O número de colunas de cada linha será o valor máximo definido para a ramificação da MHE. Vamos citar a matriz como LxC (linha x coluna)
4. Esta matriz LxC é inicializada aleatoriamente com valores 0 e 1
5. Para uma representação com valores máximos de Profundidade e Ramificação iguais a 2 (2,2), a matriz cromossômica que a representaria seria de 3 linhas e 2 colunas (3x2), conforme abaixo:

1	1
0	1
1	1

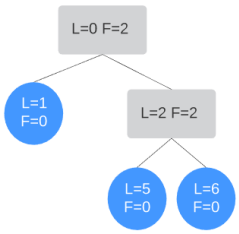
Conversão da Matriz para Representação

Para que pudessemos utilizar essa codificação no código MatLab/Octave, foi necessário efetuar a decodificação da matriz para uma estrutura de array, já considerando a quantidade de ramos e filhos os quais o algoritmo genético manipulou e aplicou os operadores genéticos. Sendo assim, foi efetuado o seguinte processo para decodificação: (manteremos o exemplo acima 2x2 como exemplo:

1. A seguinte matriz representa a seguinte árvore:



2. A primeira linha (indice 0), informa quantos filhos(nós) a raiz da MHE terá. Nesse caso, como as posições (0,0) e (0,1) da matriz são 1, representa que o nó raiz terá 2 filhos. Caso, nesse exemplo, a soma da linha resultasse em um valor <= 1, isso significaria que ele seria um especialista, ou seja, nesse caso a raiz seria uma Rede Neural Especialista, sem gating e com nenhum nó abaixo. (L = linha da matriz / F = qtde de nós filhos)



3. Caso qualquer linha tenha a somatória do valor maior que 1, significa que ela possui nós filhos, caso ela seja uma parte válida do cromossomo (explicado este ponto logo abaixo). No nosso exemplo, a linha 1 tem a somatória igual a 2. Assim, teremos que verificar em qual linha da matriz esses nós filhos estão representados. Para isto, estou utilizando a seguinte fórmula:

$$LF = (L * NC) + C + 1$$

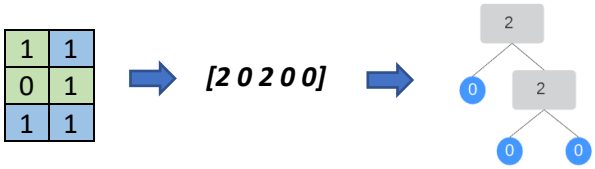
4. Onde, LR é a linha onde está representado os nós filhos, L é a linha onde esta o nó pai, NC é o número de colunas (ramificações) da matriz, C é a coluna onde está o nó pai. Assim, seguindo nosso exemplo, para descobrirmos onde estão os nós filhos da posição (0,0) de nossa matriz -> LR = (0 * 2) + 0 + 1 => 1, ou seja, os nós filhos referentes ao nó pai da posição (0,0) estão na linha 1 (em nossa matriz, pintamos de verde). O mesmo ocorre para a posição (0, 1) => (0 * 2) + 1 + 1 =>, que têm os nós filhos na linha 2.
5. Caso o resultado deste cálculo (LR) para nós filhos de uma determinha linha seja maior que o número máximo de linhas definidos na matriz, eles são considerados especialistas.

Com estes cálculos definidos acima, foi possivel definir o seguinte algoritmo, o qual utilizo para decodificar a matriz para um array que é enviado ao código MatLab/Octave.

```
    def calcular_arvore(self, matriz, linha, resultado):
        if(linha >= len(matriz)): # Condição de parada da recursão. Se for o último level.
            resultado.append(0)
            return

        soma_linha = np.sum(matriz[linha])
```

```
        if soma_linha <= 1: # Se for <= 1, então é especialista, ou seja, não tem mais de 1
filho.
            # Criar EXPERT
            resultado.append(0)
        else:
            qtd_coluna = len(matriz[0])
            resultado.append(soma_linha)
            for coluna in range(soma_linha): # Numero de colunas
                linha_ref = (linha * qtd_coluna) + coluna + 1
                self.calcular_arvore(matriz, linha_ref, resultado)
```

A matriz abaixo se transforma no seguindo array:



Com essa representação, conseguimos representar qualquer tipo de árvore de MHE.

Para que essa implementação fosse possível, também foram ajustados 2 arquivos do código MatLab/Octave:

hmeTest.m

```
function [hme_fit, yexp, z]= hmeTest(alturamax, larguramax, arquitetura)
    depth=alturamax; % Profundidade
    factor=larguramax; % fator de ramificação
    nexperts=factor; % fator de ramificação
    X = load('serie1_lag.txt');
    Y = load('serie1_lag_y.txt');
    global index_gating;
    index_gating = 1;
    [N,dim]=size(X); %J com bias
    [N,k]=size(Y);
    hme = hmeCreate(depth,nexperts,dim,k, arquitetura ); %cria a estrutura
```

hmeCreate.m

```
function [hme] = hmeCreate(levels,b,d,k, arquitetura) % MICHEL_ARQUIT
% INPUTS
% levels  number of levels (>=0) in hierarchy
% b     branching factor
% d     dimensionality of data
% k     number of outputs
%
% OUTPUTS
% hme     HME model

global index_gating
```

```
ind_arq = index_gating;
b = arquitetura(ind_arq);
index_gating = ind_arq + 1;

if levels == 0 | b <= 1,
    hme = hmeCreateExpert(d,k); % cria os especialistas
else

  children = cell(1,b); %cria os filhos

  for i = 1:b,
    if (levels-1)==0
        % disp(sprintf('\n****************************************')) % MICHEL
        % disp(sprintf('Rede Especialista do ramo %d',i))       % MICHEL
        % disp(sprintf('****************************************')) % MICHEL
    end
    children{i} = hmeCreate(levels-1,b,d,k, arquitetura); % MICHEL_FOLHAS

  end
  % disp(sprintf('\n****************************************'))% MICHEL
  % disp(sprintf('Rede Gating do nivel %d',levels))       % MICHEL
  % disp(sprintf('****************************************')) % MICHEL
  hme = hmeCreateMixture(children,d);
end
```

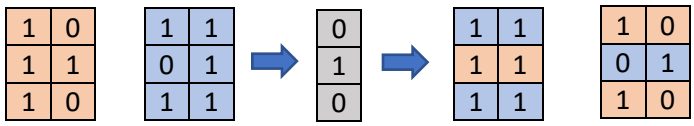
Crossover(recombinação) e Mutação utilizadas

Como a codificação aplicada neste cenário é diferente das que utilizamos até o momento (matriz ao invés de vetor/array), foi necessário aplicar um outro tipo/forma de crossover, apesar do conceito central ter sido mantido (recombinaação de pai1 e pai2 para gerar filho1 e filho2).

Neste experimento foi utilizado o crossover por linha da matriz e baseado em uma máscara gerada aleatóriamente, com valores entre 0 e 1, tendo esta máscara a dimensão de 1 coluna e com a quantidade de linhas iguais ao da matriz do cromossomo dos individuos (pais).

Resumidamente, o crossover é feito da seguinte maneira:

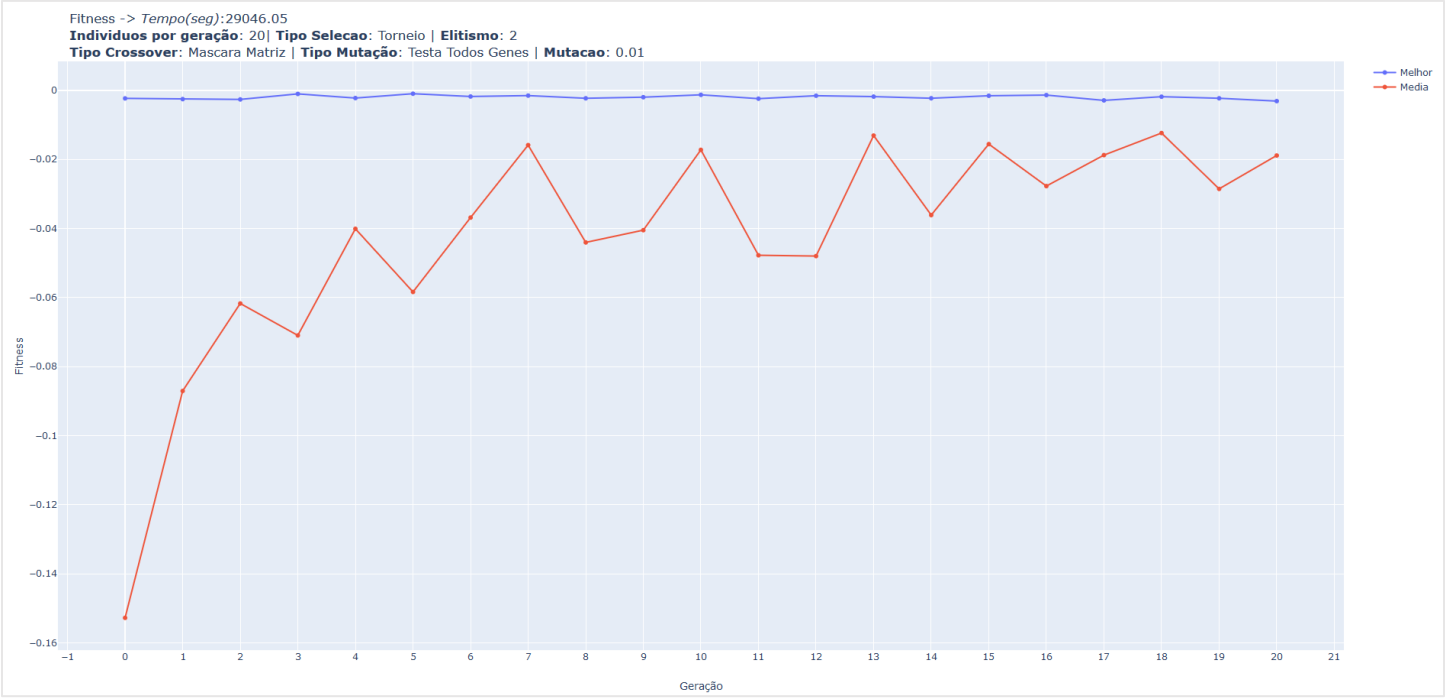
1. Após os pais serem selecionados, uma máscara é gerada com valores 0 e 1 aleatoriamente
2. Cada linha da mascara é percorrida e, caso o valor contido na linha *i* seja 1, a linha *i* do pai1 é inserida na linha *i* do filho 1 e a linha i do pai2 é inserida na linha *i* do filho 2 (filho1[i] = pai1[i] / filho2[i] = pai2[i]). Caso o valor da mascara seja 0, o inverso acontece (filho2[i] = pai1[i] / filho1[i] = pai2[i]). Abaixo, o exemplo:



A mutação foi aplicada gene a gene, ou seja, cada linha/coluna foi percorrida, caso o valor randômico fosse menor que a taxa de mutação estabelecida, a mutação era aplicada a este gene, mudando de 0 -> 1 ou de 1-> 0.

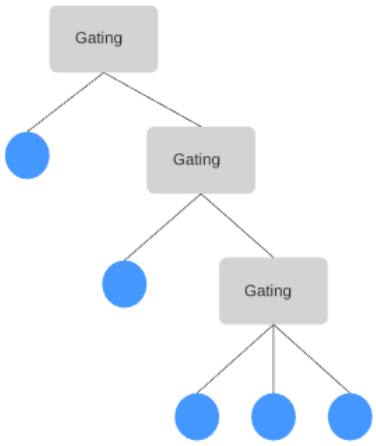
Abaixo, alguns experimentos realizados, limitados em profundidade e ramificação em 3, e seus respectivos gráficos:

Gerações: *20* ***| Individuos por Geração:*** *20* ***| Tipo Seleção:*** *Torneio* ***| Crossover****: Máscara* ***| Mutação:*** *Verifica Todos os Genes* ***| Mutação:*** *0.01*



Dados2

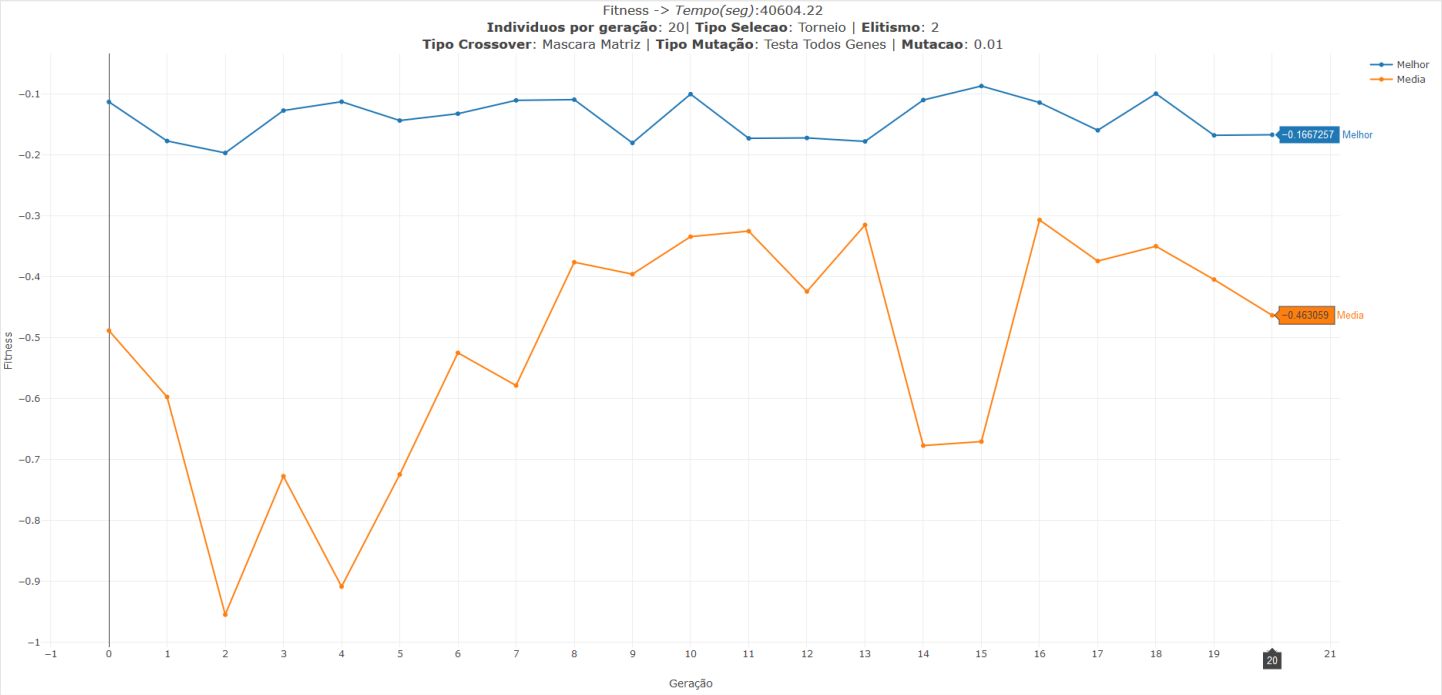
Nesta execução, como podemos ver, usando MHE com arquiteturas totalmente variáveis (limitadas a um limite de profunidade e ramificação, que neste caso foram ambos 3), o melhor indivíduo foi mais constante em sua nota durante as gerações, variando entre -0.003 e -0.0009 de Fitness (LLI). Dentre todas as gerações, o melhor indivíduo foi na geração 6, com a arquitetura decodificada em [2 0 2 0 3 0 0 0]. Esta arquitetura MHE esta representada abaixo:



Conforme informado na página a seguir, esta estratégia também foi executada sobre a Serie 1, série temporal dos trabalhos anteriores;

A execução gerou o seguinte gráfico:

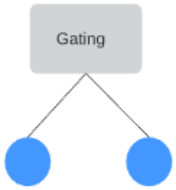
***Gerações:** 20 | **Individuos por Geração:** 20 | **Tipo Seleção:** Torneio | **Crossover**: Máscara | **Mutação:** Verifica Todos os Genes | **Mutação:** 0.01*



Serie1

Esta série tem um comportamento diferente da série dos demais experimentos acima, que é mais constante, o acabou gerando, conforme acima, uma variação maior entre a média e os melhores individuos entre as gerações.

A arquitetura que obteve o melhor Fitness (likelihood) dentre as 20 gerações foi a com decodificação [2 0 0], obtendo o Fitness de -0.086, na geração 16. A arquitet ura representasa é:



Características Técnicas das Execuções

Ao serem iniciadas as 1º execuções deste trabalho, foi percebido que a demora para cada iteração e avaliação de cada indivíduo utilizava um tempo onsiderável no que, não seria possível executá-las em um tempo hábil para a entrega do trabalho. Para isso, foi alugado um máquina na AWS, com 16 núcleos, o qual também verifiquei que não houve uma melhora siginificativa.

Tendo isso em vista, ao analisar o motivo de não ter havido melhor, identifiquei que, apesar de 16 núcles, a somente 1 núcleo da VM era utilizado, indicando que a programação não estava paralelizada.

Com a intenção de melhorar a performance, foi alterado o Algoritmo Genético, onde a cada geração, a avaliação dos indivíduos dentro da mesma geração, são executadas de forma paralela, pois são independentes. Após esta

modificação, foi possível obter uma melhora de performance e de execução de mais de 50%, reduzinho praticamente pela metade o tempo gasto.

Sendo assim, os experimentos foram executados em 3 tipos de máquinas distintas:

- AWS – Windows Server 19 – 16 núcleos – 32RAM
- Azure – Windows 10 Pro – 8 núcleos – 16 RAM
- Notebook – Windows 10 Pro – 4 núcles – 16 RAM

Outros experimentos pretendidos de serem colocados não terminaram a tempo da entrega do trabalho (mais execuções sobre a Série 1).

Possíveis futuros trabalhos

Utilizamos para estes estudos apenas uma função Fitness, baseada no Likelihood. Eu pretendia também, neste estudo, aplicar o EQM da MHE como Função de Fitness, mas com o intuito de minimizá-la (indivíduos com menores valores seriam os melhores).

Também pode ser aplicado a variação de neurônios dos gatings e especialistas, como também os tipos dos gatings e especialistas (linear, mlp, etc)

Também, neste trabalho os tamanhos dos arrays e matrizes contém a quantidade de bits máxima que a maior rede, dentro do limite estabelecido, pode ter. Uma outra evolução seria deixar estas codificações dinâmicas, fazendo com que as codificações de bits não contenham “genes adormecidos”, que não representam e nem influenciam aquele individuo, apesar de serem úteis no momento do crossover(recombinação).

Conclusão

Este trabalho foi de grande valia pois foi possível entender e avaliar, através dos diversos experimentos, o quão interessante e eficiente pode ser a integração de Algoritmos Genéticos e MHE, para que, dentro de uma grande quantidade de combinações (os quais não seja possível percorrer por força bruta, por exemplo), seja evoluída uma população e identificada uma boa, ou talvez até ótima, arquitetura de MHE para um determinado problema.

Foi possível notar também que, nas séries utilizadas neste trabalho, MHE com arquiteturas menores se comportaram e obtiveram um resultado melhor do que arquiteturas com muitos niveis de profundidade e ramificação. É interessante pois, geralmente quem está iniciando nesta área, parte do pressuposto que, quanto mais gatings, especialistas, niveis e ramos, a rede se comportará melhor, o que, nem sempre é o correto, como pudemos exibir neste trabalho.

Foi também identificado que é possível melhorar a performance de um algoritmo genético, principalmente neste caso onde existe grande processamente e reavaliação constante das MHE. Neste caso, para exeprimento, foi preservado o valor da primeira execução de uma determinada arquitetura MHE, para que não fosse necessário reexecutá-la novamente em futuras gerações, caso ela fosse gerada. Isto necessita ser minuciosamente analisado pois, para alguns problemas, será necessário efetuar esta reavaliação dos indivíduos, e não será interessante essa melhoria de performance comparado com a qualidade do resultado final. Neste exemplo, poderiamos evoluir e, ao invés de guardarmos uma vez e nunca mais o reexecutarmos(reavaliarmos) o indivíduo até o final das gerações, poderiamos de tempos em tempos (colocar um contador para cada arquitetura armazenada) reexecutarmos essa estrutura e, por exemplo, tirarmos a média das execuções realizadas. Novamente, seria necessário uma análise mais minuciosa para verificar o verdadeiro impacto dessas soluções propostas neste parágrafo.

Também, como o intuito deste experimento foi manipular a arquitetura da MHE, não inclui bits (genes) para variação dos neurônios de cada especialista da rede e também nem para o tipo de cada especialista/gating. Dada as

estratégias citadas neste estudo, seria totalmente viável incluirmos bits que os representassem (tipo de rede, ex.: linear, mlp, etc, e qtde de neuronios, ex: entre 1 a 16). Por isso deixamos definidos os tipos de todas redes e gating (ex.: MLP) e a quantidade de neuronios para todos os especialistas (ex.: 3). Os demais parâmetros da MHE, tiveram iteração configurada para 10.