# Proposta de Algoritmo Genético aplicado em Redes Neurais usando Keras

Carlos Eurico Galvão Rosa (UFPR) cegalvao@ufpr.br
Deise Maria Bertholdi Costa (UFPR) deise@ufpr.br

**Resumo:**
Dimensionar um modelo de Rede Neural, com todo seu detalhe de arquitetura e a aplicação de técnicas de melhoria no método pode não ser um trabalho simples quando cresce o número de possibilidades. O uso de metaheurísticas tem obtido bons resultados em otimização e é suficientemente versátil para ser aplicado em um problema de construção de outra metaheurística. Este artigo traz uma possibilidade de hibridização para obter uma Rede Neural com boa performance escolhendo seus parâmetros não de forma arbitrária, mas com uso de outra metaheurística também importante: Algoritmos Genéticos.
**Palavras chave**: Otimização, Algoritmo Genético, Redes Neurais, Keras.

# Propose of Genetic Algorithm applied in Neural Networks using Keras

**Abstract**
Scaling a Neural Network model with all its architectural detail and applying method improvement techniques may not be a simple job as the number of possibilities grows. The use of metaheuristics has been successful in optimization and is versatile enough to be applied to a problem of building another metaheuristic. This article provides a possibility of hybridization to obtain a good performing Neural Network by choosing its parameters not arbitrarily, but using other equally important metaheuristic: Genetic Algorithms.
**Key-words:** Optimization, Genetic Algorithms, Neural Network, Keras

## 1. Introdução

Muitas vezes a resolução de um problema demanda alto custo computacional, especialmente quando o número de possíveis soluções é calculado através de produto da quantidade de valores que cada variável pode assumir, ou ainda, quando envolve combinatória. Esse é o caso da biblioteca Keras, uma poderosa ferramenta que facilita a construção de modelos de Redes Neurais aplicáveis a diversos tipos de problemas. Uma boa configuração da rede pode obter resultados com maior precisão, melhor performance ou menor tempo computacional. Considerando apenas as possibilidades de funções de ativação, usadas nos neurônios das camadas da rede, inicializadores, função objetivo e otimizadores, são quase vinte mil possíveis combinações distintas de modelos. Levando em conta que também é necessário definir outros parâmetros como quantidades de camadas, tamanho das camadas, quantidades de épocas de treinamento, possibilidades de *dropout* e outras variáveis, esse número de

possibilidades é ainda maior. Testar todas as possíveis combinações para obter a solução ótima de forma deterministica, levando em conta os gastos computacionais da própria rede, pode ser um procedimento custoso.

Encarando o dimensionamento e as configurações do modelo de Redes Neurais com uso de Keras como um problema a ser otimizado, a aplicação de outra metaheurística desponta como uma viável forma de obtenção de boas soluções deste caso. Metaheurísticas, para Glover e Kochenberger (2003), são métodos que combinam mecanismos de melhorias locais com estratégias de alto nível em busca de melhores soluções para problemas complexos.

Para a presente proposta, foi escolhido usar os Algoritmos Genéticos para gerenciar a busca por melhores configurações de Redes Neurais. O presente artigo, que é parte inicial de pesquisa sobre o tema, discorre brevemente sobre os métodos heurísticos e técnicas utilizadas na Seção 2. Fazendo dessa composição de métodos heurísticos um híbrido, a seção 3 relata o procedimento utilizado para a operacionalização deste híbrido, onde um conjunto de Redes Neurais se torna uma população de Algoritmo Genético, submetida aos procedimentos desta técnica. Ao fim são apresentadas as indicações de continuação do estudo e considerações do tema.

## 2. Métodos utilizados

Apresenta-se nesta seção um pouco da teoria que fundamenta os métodos utilizados para a hibridização: as metaheurísticas Algoritmos Genéticos (AG) e Redes Neurais Artificiais (RNA) bem como o *framework* Tensorflow e a biblioteca Keras.

### 2.1 Algoritmos Genéticos (AG)

Inspirados em modelos biológicos, os AG são parte do grupo de metaheurísticas ditas populacionais, tendo um grupo de possíveis soluções que são aprimoradas através de processos, como a seleção, o *crossover* e a mutação, que mimetizam a evolução biológica através de gerações. Mikuska (2015), baseando-se em Goldberg (1989), apresenta um paralelo entre a terminologia oriunda da biologia, dita natural, e os elementos operacionalizados pelo AG aqui reproduzido na Tabela 1

| Natural | AG |
|---|---|
| População | Conjunto de Soluções |
| Indivíduo | Solução do Problema |
| Cromossomo | String que compõe a solução do problema |
| Gene | Característica |
| Alelo | Valor da Característica |

Fonte: Mikuska (2015) adaptado de Goldberg (1989)

Tabela 2 – Comparação entre terminologia natural e AG

Cada indivíduo que compõe a população inicial é gerado por algum método randômico, guloso ou não. Criada a população inicial, o AG passa a sua fase iterativa, sendo as principais etapas das iterações do AG a avaliação; a seleção de reprodutores; a geração de novos indivíduos por meio de operações e atualização da população. A figura 1 apresenta um fluxograma que destaca essas etapas.

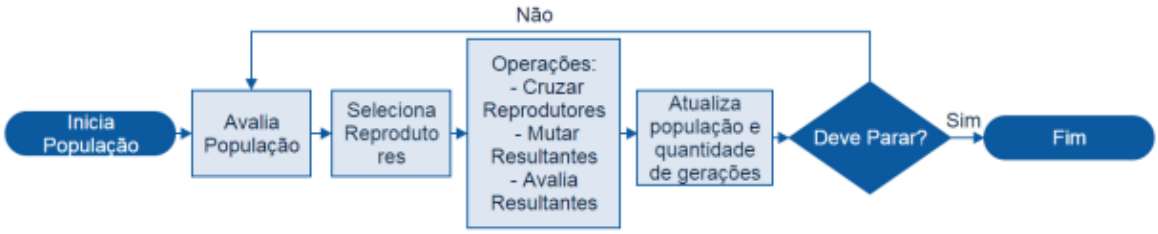


Figura 1 – Estrutura e funcionamento de AG
Fonte: Mikuska (2015)

Na etapa de avaliação cada indivíduo é submetido a uma função de qualidade chamada *fitness*, ou função de aptidão, que pode ser custo (a ser minimizado) ou taxa de acerto (a ser maximizada). São estabelecidos critérios para a seleção de indivíduos, ditos geradores, que passarão por operações que gerarão novos indivíduos, chamados gerados. A figura 2 apresenta um exemplo hipotético de cromossomos de dois indivíduos distintos passando pelo processo de *crossover*. Cada cromossomos apresenta oito genes (os retângulos da figura) e cada gene tem uma informação, o alelo. Aplicado um ponto de corte em cada um deles, novos cromossomos são gerados a partir da união das respectivas partes dos anteriores após a troca. No caso, a parte inicial de um cromossomo é unida à parte final do outro. Nota-se que, após este processo, a população ganha dois indivíduos gerados, que terão os cromossomos resultantes do *crossover*, mantendo os geradores.

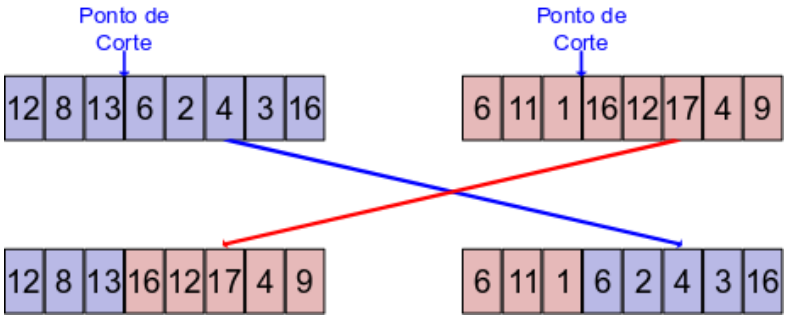


Figura 2 – Processo de *crossover*
Fonte: Os autores

É calculado o valor de *fitness* de todos os indivíduos gerados, visto que na etapa de avaliação já havia sido calculado dos anteriores. A seleção de indivíduos que formarão a próxima geração pode ser exclusivamente com os indivíduos de melhor *fitness* dentro do contexto do problema, chamada de seleção elitista. Outra forma de seleção envolve manter algum percentual de soluções que não estejam dentre as de melhor *fitness* a fim de que se tenha diversidade de soluções na população e sejam evitados possíveis ótimos locais. Terminada a nova geração é verificado o atendimento aos critérios de parada, que pode ser a existência de algum indivíduo com *fitness* próximo do desejado, dentro de uma tolerância, ou uma quantidade pré-estabelecida de gerações para evitar *loop* infinito.

## 2.2 Redes Neurais Artificiais (RNA)

De modo semelhante aos AG, as RNA também são bioinspiradas, mimetizando o funcionamento de células cerebrais. Haykin (2001) define as RNA como

> Uma rede neural é um processador maciçamente paralelamente distribuído constituído de unidades de processamento simples, que têm a propensão natural para armazenar conhecimento de unidades de processamento simples, que têm a

propensão natural para armazenar conhecimento experimental e torná-lo disponível para o uso. Ela se assemelha ao cérebro em dois aspectos:
1. O conhecimento é adquirido pela rede a partir de seu ambiente através de um processo de aprendizagem.
2. Forças de conexão entre neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido.

O esquema mais simples de RNA é o *perceptron* onde um neurônio recebe informações de entrada, processa com uma função de ativação e fornece um resultado de saída. A RNA aprende padrões de saída com a repetida apresentação de dados de um subconjunto de informações de entrada, chamadas entradas de treino, enquanto a validação é feita com outro subconjunto das mesmas informações, disjunto do anterior, com as entradas de teste. Uma ampliação do *perceptron* comum é o *perceptron multicamadas* (MLP, do inglês *multi layer perceptron*), ilustrado na Figura 3, com três neurônios de entrada, duas camadas, e dois neurônios de saída.

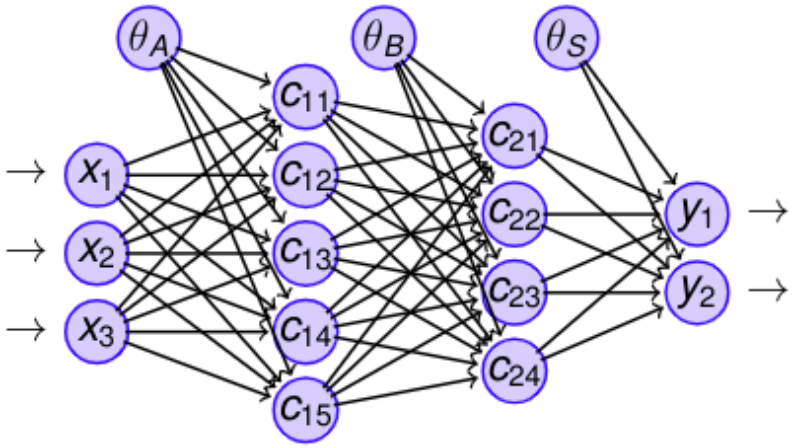


Figura 3 – Exemplo de Rede MLP
Fonte: Os autores

Na Figura 2, os neurônios de entrada estão na primeira coluna ($x_1$, $x_2$, $x_3$), os neurônios de saída na última coluna ($y_1$, $y_2$), sendo os demais pertencentes às camadas ocultas. As setas que ligam as camadas representam as conexões sinápticas, sendo cada uma com um peso específico. A primeira linha de neurônios, identificados com a letra θ, são os *bias* ou *viés* associados a cada camada, com influência nos cálculos de saída. Generalizando a representação dos neurônios como $c_{ij}$, sendo i = 0 para camada de entrada, denotando $w_{ikj}$ a conexão sináptica entre o neurônio $c_{ik}$ com o neurônio $c_{(i+1)j}$ com $0 \leq k \leq n_i$, sendo $n_i$ o número de neurônios da i-ésima camada com k=0 para o *bias* que influencia a camada seguinte, e *f* a função de ativação definida, é possível representar o cálculo do valor de um neurônio pelo somatório

$$c_{(i+1)j} = f\left(\sum_{k=0}^{n_i} w_{ikj} c_{ik}\right).$$

A inicialização dos valores de peso é feita de forma aleatória e, após calculados os valores de todos os neurônios, é medida a diferença entre o resultado gerado pela rede e o resultado esperado para o padrão informado, sendo este o *erro* da apresentação, cujo objetivo de todo o processo é minimizá-lo. Com este erro, no MLP, é possível fazer a atualização dos valores dos pesos sinápticos, considerando o algoritmo de retropropagação (*backpropagation*) com

descida do gradiente da função de ativação.

### 2.3 Tensorflow

Com o objetivo de "experimentar novos modelos, treinando-os em grandes conjuntos de dados e movendo-os para a produção" (ABADI et. al., 2016), uma equipe de profissionais do Google construiu o Tensorflow a partir de experiências com o primeiro sistema usado para treinamento de RNA, o DistBelief. Isto se deu, segundo Abadi et. al. (2016) "simplificando e generalizando para permitir que os pesquisadores explorem uma variedade maior de ideias com relativa facilidade".

Ainda conforme Abadi et. al (2016), trata-se de "um sistema de aprendizado de máquina que opera em larga escala e em ambientes heterogêneos" com "gráficos de fluxo de dados para representar a computação, o estado compartilhado e as operações que modificam esse estado." Buscou-se combinar programação de alto nível dos sistemas de fluxo de dados e na eficiência de baixo nível dos servidores de parâmetros.

### 2.4 Keras

Segundo Moolayil (2018), "Keras é uma API de rede neural de alto nível escrita em Python e pode ajudá-lo no desenvolvimento de um modelo de *Deep Learning* totalmente funcional com menos de 15 linhas de código". API (do inglês *Application Programming Interface*, Interface de Programação de Aplicativos) é um conjunto de rotinas, padrões de programação e protocolos usados no desenvolvimento e integração de *software*. Dessa forma, o uso do Keras facilita a programação de RNA automatizando diversas rotinas, suportando diversos *frameworks* de baixo nível, como o *Tensorflow*.

Um código simples usando Keras pode ser dividido em três etapas: Leitura e obtenção dos dados; Definição da Estrutura do Modelo e; Treinamento do Modelo seguido de predições. Os dados lidos podem ser de diferentes tipos, porém podem ser representados por tensores, que são generalizações n-dimensionais de vetores. Para a operacionalização, os dados lidos precisam ser tratados de forma numérica, sejam rótulos, pixels de imagens ou outro tipo de informação. Para melhorar o processo computacional, tanto Hayakin (2001) quanto Moolayil (2018) recomendam que os dados informados ao modelo já estejam normalizados em um intervalo [0,1), ou então padronizados com média zero e desvio padrão um.

Para a Estrutura do Modelo, pode-se optar pelo *Sequential*, recomendado para casos mais simples ou *Model*, para casos mais complexos, como saídas múltiplas, camadas compartilhadas ou grafos acíclicos. Em ambos os casos, o Keras usa *Layers* (camadas) onde são definidas as quantidades de neurônios, função de ativação específica para a camada, uso de *bias*, inicializadores de valores de pesos e *bias* (com zeros, uns, aleatórios de distribuição normal, aleatórios de distribuição uniforme entre outras possibilidades, inclusive com inicialização personalizada) e reguladores que permitem aplicar penalidades em parâmetros de camadas ou suas atividades durante a otimização, incorporadas na função de perda. A *layer* principal é dita *Dense*, que é uma camada na qual todos os neurônios nela são conectados com as camadas próximas. A fim de evitar um sobredimensionamento (*overfitting*) de neurônios da camada, Srivastava et. al (2014) sugerem o uso de *Dropout*: selecionar aleatoriamente um percentual de neurônios a ser ignorado no processamento em uma execução. A figura 4 ilustra um modelo de rede completa e uma rede com o *Dropout*. O Keras contém uma função específica para a aplicação de *Dropout* em seus modelos.

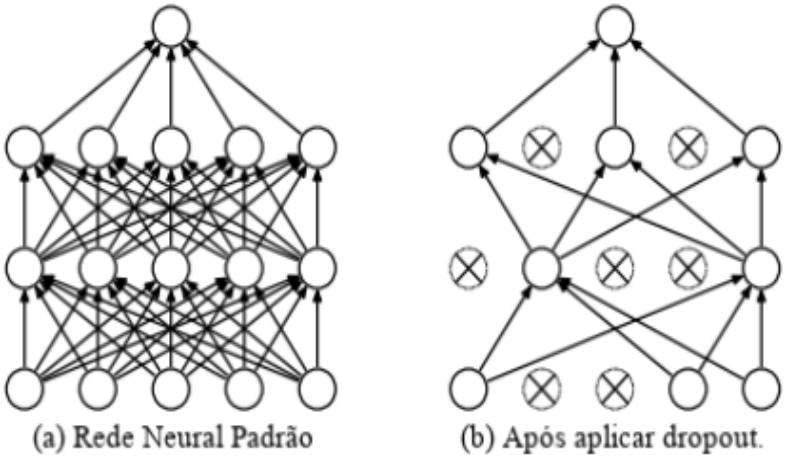


Figura 4 – Aplicação de dropout em rede.
Fonte: Srivastava et. al (2014)

Para outras situações específicas, como redes recorrentes, convolucionais, agrupamento de dados entre outros, existem métodos embutidos no Keras que podem ser utilizados. Além disso, é possível que o pesquisador crie sua própria camada personalizada.

A função de perda, mencionada ao citar os reguladores e que também pode ser conhecida como função objetivo, é o meio que o modelo "entende" se está aprendendo o padrão como se espera e depende da forma dos dados de saída do problema. Em saídas numéricas, esta função pode ser Erro Médio Quadrático, Erro Médio Absoluto, Erro Médio Logarítmico, Erro Médio Absoluto Percentual, entre outras. Em saídas de categorias, são possíveis as funções de Entropia Cruzada Binária, nos casos Verdadeiro/Falso, ou Entropia Cruzada Categórica para Sim/Talvez/Não. Semelhante à função de perda, o Keras tem as métricas, para medir a acurácia do modelo, com a diferença que estas não são levadas em conta na fase de treinamento.

Finalizando a construção do modelo, é necessário definir qual será o otimizador utilizado no modelo. Tanto a função de perda quanto o otimizador são parâmetros obrigatórios no Keras. A escolha do otimizador define como a função de perda será minimizada e a taxa de acertos do modelo aumenta. Dentre diversos otimizadores disponíveis no Keras, os algoritmos mais comuns são o de Descida Estocástica do Gradiente (*Stochastic Gradient Descent* - SGD), *Adadelta*, de Zeiler (2012) e *Adam*, de Kingma e Ba (2015), com suas variantes *Adamax*, também descrito por Kingma e Ba (2015) e *Nadam*, apresentado por Dozat (2016).

Na última das etapas citadas, é feito o treinamento do modelo. Para isso são usados os métodos *compile* para configurar o modelo; *fit* para treinar o modelo em uma quantidade definida de épocas; *evaluate* que retorna os valores de métricas e função de perda e *predict* para gerar saídas com previsões a partir de conjuntos de entrada com o modelo testado.

## 3. Procedimento de Hibridização

Na seção anterior foram exibidas as ferramentas utilizadas para a hibridização, em especial a diversidade de parâmetros disponíveis para a construção do modelo com o Keras. Nesta seção é abordada a forma que o AG será aplicado na construção dos modelos e busca dos melhores resultados com as redes, iniciando com as características do problema, a forma do tratamento do problema com a linguagem AG e como se dão as operações de AG no modelo Keras.

### 3.1. Caracterização do modelo e opções disponíveis

Para o estudo inicial da hibridização fica definido um modelo padrão de rede, do qual se originará a estrutura do indivíduo e seus cromossomos. A tabela 2 complementa a tabela 1, inserindo naquele mesmo contexto os elementos usados no Keras e fazendo a relação entre as terminologias dos três contextos.

| Natural | AG | Keras |
|---|---|---|
| População | Conjunto de Soluções | Conjunto de modelos de RNA |
| Indivíduo | Solução do Problema | Um modelo de RNA |
| Cromossomo | String que compõe a solução do problema | Etapa da RNA |
| Gene | Característica | Item do modelo |
| Alelo | Valor da Característica | Parâmetro do item |

Fonte: Os Autores, adaptado de Mikuska (2015)

Tabela 2 – Associação termos naturais AG e Keras

Assim, é possível descrever os procedimentos de operacionalização do AG, estabelecendo relação entre a linguagem natural e os parâmetros próprios do modelo Keras, explicitados na seção anterior. Para a execução do modelo, algumas informações são necessárias para balizar a criação da população e o modo de funcionamento e operação do AG. Neste sentido, a tabela 3 apresenta nomes, definições e/ou possíveis valores destes parâmetros.

| Parâmetro | Descrição |
|---|---|
| Tam_Populacao | Quantidade de indivíduos de uma população por geração. |
| Percent_Sobreviver | Percentual de melhores indivíduos que passam para a geração seguinte. Valor entre 0 e 100. |
| Qtde_Geracoes_Max | Máximo de execuções do AG, um dos critérios de parada. |
| Tolerancia_Parada | Taxa de acerto mínima que pode interromper o AG antes de atingir Qtde_Geracoes_Max. |
| Arquivo_Entradas | Arquivo com dados a serem lidos e processados pela rede |
| Dimensao_Entradas | Dimensão dos valores lidos para as primeiras camadas de cada rede. |
| Colunas_Entrada | Identificação das colunas de dados que serão usadas como dados de entrada |
| Colunas_Desejo | Identificação de colunas com valores desejados de saída |
| Qtde_operacoes | Quantidade de operações de reprodução a serem realizadas no processo do AG |
| Tam_Treino | Quantidade de dados que formam o conjunto de treinamento da RNA, deixando os demais dados do Arquivo_Entradas como conjunto de testes. |
| Arquivo_Saida | Nome do arquivo que será gerado. Opcional. |
| Metrica_Comparacao | Função que será aplicada em todos os indivíduos a fim de rankear os indivíduos. |

Fonte: Os Autores

Tabela 3 – Parâmetros iniciais do AG

Definidas as operações de população, é preciso estabelecer as características do indivíduo. Para o modelo aqui proposto foram definidos três cromossomos: Entrada, Camadas e Treino. Isso se baseia nas etapas descritas do Keras e porque uma eventual mistura de informações entre etapas poderia gerar novos indivíduos inviáveis. Os cruzamentos do AG serão feitos apenas entre os genes dos respectivos cromossomos, não permitindo que imagens relativas à entrada de dados no modelo, por exemplo, não venha a ser omitida por uma característica de outra etapa. Apresentam-se, na Tabela 4, cromossomos acima definidos e seus respectivos genes de um indivíduo desta proposta.

| Cromossomo | Gene | Descrição |
|---|---|---|
| Entrada | Conjunto_Treino | Entradas que serão usadas para o treinamento da rede. |
| Entrada | Conjunto_Teste | Entradas que serão usadas na validação da rede e, necessariamente não fazem parte do Conjunto_Treino |
| Entrada | Qtde_Camadas | Quantidade de Camadas da Rede que terá o modelo |
| Camadas | Qtde_Neuronios | Quantidade de Neurônios da Camada em questão |
| Camadas | F_Ativacao | Função de ativação dos neurônios da camada |
| Camadas | Dropout | Taxa de dropout a ser aplicada na camada |
| Treino | Optmizer | Otimizador escolhido para a rede. Exemplos |
| Treino | Loss | Função objetivo do modelo |
| Treino | Metrics | Métricas do modelo |
| Treino | Epochs | Número de épocas de treinamento |

Fonte: Os Autores

Tabela 4 – Características do indivíduo

Cada rede será executada de acordo com as características apresentadas pelos genes do indivíduo. A avaliação de cada indivíduo se dará pela avaliação do modelo com os dados do Conjunto de Teste, sendo o *fitness* do indivíduo o resultado da métrica de comparação estabelecida para a população, permitindo o rankeamento dos indivíduos.

### 3.2. Seleção de reprodutores e funcionamentos das operações de AG

Os reprodutores serão selecionados pelo sistema de "roleta", onde os indivíduos de melhor posição no ranqueamento terão maior chance de sorteio, Por exemplo, em uma população de dez indivíduos, o indivíduo de melhor *fitness* ocupa a primeira posição e terá dez chances de ser sorteado, o segundo colocado terá nove chances e assim sucessivamente. Em caso de empate em alguma colocação, os indivíduos empatados terão chance igual.

As operações a serem feitas poderão ser aleatórias ou dirigidas, dependendo da forma que se desejar conduzir a pesquisa. É possível a geração de novos indivíduos através de:

- Mutação: Um alelo tem seu valor alterado para outro valor possível dentro do contexto;
- *Crossover* de cromossomos: Como cada indivíduo tem três cromossomos, é possível gerar novos indivíduos através da combinação de cromossomos inteiros de dois ou mais geradores;
- *Crossover* de genes com um ponto de corte: semelhante ao exemplificado na figura 2 e explanado naquela seção;
- *Crossover* de genes com dois pontos de corte: os genes que estiverem entre os pontos de corte de um cromossomo são unidos aos genes que não estiverem entre os cortes no outro cromossomo.

As operações podem ser aplicadas isoladamente ou de forma combinada, atuando em mais de um pedaço de configuração do indivíduo.

### 3.3. Avaliação, formação de nova geração e verificação do critério de parada

A cada novo indivíduo gerado é calculado seu *fitness* e este indivíduo é posicionado em ranqueamento junto com os já existentes, sem alteração nas chances de sorteio de próximos geradores durante o processo. Adota-se o percentual de sobrevivência (Percent_Sobreviver) de modo que, para Percent_Sobreviver = n < 100%, n% das posições na nova geração (considerando truncamento de números fracionários de indivíduos) são ocupadas pelos indivíduos melhor posicionados no ranqueamento. As vagas restantes são ocupadas por meio de sorteio uniforme dos demais indivíduos, sendo descartados os indivíduos excedentes.

Caso a nova geração atenda algum dos critérios de parada pré-estabelecidos, o AG é encerrado e o indivíduo de melhor colocação no último ranqueamento é tomado como solução.

## 4. Considerações e prosseguimento dos estudos

A união de dois procedimentos metaheurísticos pode trazer bons resultados levando em conta seus pontos positivos. O uso de AG no modelo do Keras se mostra bastante natural onde algumas mutações podem causar grandes perdas ou consideráveis melhoras na precisão da resposta predita por uma RNA. E a facilidade da criação e manipulação do modelo Keras contribuem com esse estudo. Dando continuidade ao estudo, será necessário confrontar essa proposta com questões de eficiência e agilidade na obtenção da resposta e o quanto este procedimento contribui em termos de acertos preditivos do modelo. Também é possível abordar a questão dos direcionamentos de processos de geração de novos indivíduos, mensurando os impactos que cada tipo de operação do AG gera na RNA.

## Referências

**ABADI, M. et al.** *TensorFlow: A System for Large-Scale Machine Learning*. In:12TH USENIX Symposium on Operating Systems Design and Implementation (OSDI16). Savannah, GA: USENIX Association, nov. 2016. p. 265–283. Disponível em: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.

**DOZAT, T.** *Incorporating Nesterov momentum into Adam*. In: 4th International Conference on Learning Representations, ICLR, 2016. Disponível em: <http://cs229.stanford.edu/proj2015/054_report.pdf>.

**GLOVER, F.; KOCHENBERGER, G. A (eds).** *Handbook of Metaheuristics*. Boston: Kluwer Academic Publishers, 2003.

**GOLDBERG, D. E.** *Genetic algorithms in search, optmization, and machine learning*. Massachusetts: Addison-Wesley Publishing Company Inc., 1989.

**HAYKIN, S.** *Redes Neurais: Princípios e práticas*. 2ª ed. Porto Alegre: Bookman, 2001.

**KINGMA, D. P.; BA, J.** *Adam: A Method for Stochastic Optimization*. In: 3rd International Conference on Learning Representations, ICLR, 2015. Disponível em: <https://arxiv.org/pdf/1412.6980v8.pdf>.

**MIKUSKA, M. I. S.** *Uma proposta baseada em Algoritmo Genético para o problema Timetable escolar compacto* Dissertação (mestrado) - Universidade Federal do Paraná, Programa de Pós-Graduação em Métodos Numéricos em Engenharia, 2015. Disponível em: <https://acervodigital.ufpr.br/handle/1884/41889>

**MOOLAYIL, J.** *Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python* 1ª ed. Apress, Berkely, CA, 2018.

**SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R**. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. In: Journal of Machine Learning Research. no. 15, 2014. p.1929-1958. Disponível em: <http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf>.

**ZEILER, M. D**. *ADADELTA: An Adaptive Learning Rate Method*. 2012. Disponível em: <https://arxiv.org/pdf/1212.5701.pdf>.