Treinamento de redes neurais como um processo evolutivo

LUCAS OLIVEIRA DAVID *

*Ciência da Computação - Mestrado
E-mail: lucas.david@drexel.edu

***Resumo* – O treinamento de redes neurais ocorre, muitas vezes, através do algoritmo *gradient descent* utilizando *backward propagation of errors* (*gradient descent*). Entretanto, é possível encontrar autores que sugerem que este mesmo seja abordado como um processo evolutivo. Este trabalho tem como objetivo apresentar resultados empíricos obtidos do treinamento de redes neurais combinado com algoritmos genéticos. Observou-se que redes neurais criadas no processo evolutivo obtinham pontuações similares às redes treinadas convencionalmente, embora o processo evolutivo tenha exigido considerável tempo de execução. Também observou-se que redes geradas pelo processo evolutivo apresentavam fronteiras de decisão suaves, quando comparadas com o modelo gerados pelo *gradient descent*. Todos os artefatos utilizados para construir e treinar as redes, bem como os resultados obtidos podem ser encontrados como o exemplo github.com/lucasdavid/artificial/master/examples/nn-training na biblioteca github.com/lucasdavid/artificial.**

***Palavras-chave* – Aprendizado de máquina, algoritmos genéticos, redes neurais.**

I. INTRODUÇÃO

Vários algoritmos, técnicas e métodos foram desenvolvidos no aprendizado de máquina e áreas similares a fim de criar modelos capazes de generalizar sinais e padrões em conjuntos da dados. Dentre estes, destacam-se as redes neurais artificiais (RN). Sendo suficientemente genéricas, podem ser empregadas em ambas aprendizagens supervisionada e não-supervisionada, em tarefas das mais diversas, como classificação, regressão, clusterização, redução dimensional etc. Por simplicidade, este trabalho se foca em tarefas supervisionadas de classificação. Neste contexto, a aprendizagem de uma RN se dá pelo processo de treinamento, onde lhe é apresentado um conjunto de dados e seus parâmetros são alterados a fim de ajustar-se àquele conjunto.

Naturalmente, a acurácia de um modelo de aprendizado depende fortemente de seu treinamento, sendo esse um recorrente objeto de pesquisa na área de aprendizado de máquina. Comumente, o treinamento de uma RN classificadora ocorre através do algoritmo *gradient descent*, um método local que busca minimizar iterativamente uma função de erro L que avalia a diferença entre o sinal da rede com o sinal supervisionado associado à cada amostra no conjunto de dados. Dado a natureza deste problema (optimização), é possível considerar uma abordagem alternativa ao *gradient descent*: o emprego de algoritmos genéticos. Tal abordagem não é recente. Dentre vários autores, destacam-se Montana e Davis, que em 1989 descreveram um experimento onde algoritmos genéticos foram empregados com sucesso no treinamento de redes [1]; em 1994, Koehn menciona múltiplas abordagens criadas por diversos pesquisadores em como representar RNs como indivíduos do processo evolutivo, bem como diversos experimentos da acurácia de redes geneticamente selecionadas [2].

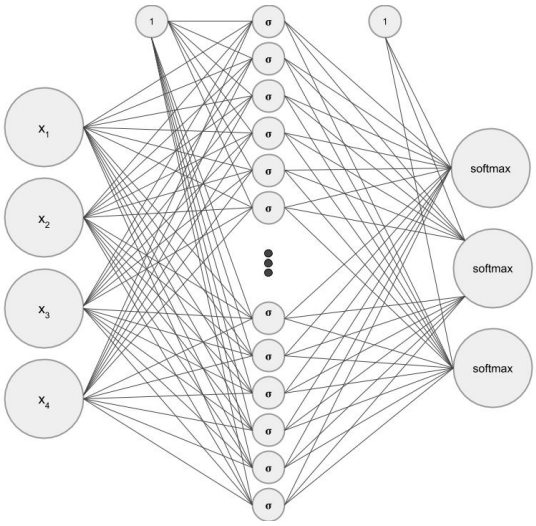
Este trabalho encontra-se organizado da seguinte forma: a seção 2 apresentará fundamentos necessários para o treinamento e evolução de redes neurais. A seção 3 apresenta uma modelagem do treinamento de RNs como um problema de optimização genética. A seção 4 lista resultados empíricos obtidos a partir da modelagem proposta e os discute brevemente. Finalmente, as conclusões são apresentadas na quinta e última seção.

II. FUNDAMENTOS

A. Redes neurais artificiais

Uma rede neural artificial é um modelo inspirado na estrutura de uma rede neural biológica [3]. Figura 1 exemplifica abstratamente uma rede neural, composta por três camadas com 1, N e 3 unidades, respectivamente.

Figura 1. Ilustração de uma rede neural com três camadas, com função de ativação σ na segunda camada e $softmax$ como função de ativação na terceira.



Embora redes neurais sejam corriqueiramente definidas como digrafos com valores reais associados aos vértices e arestas [1], será adotado aqui uma definição alternativa mais

próxima à um indivíduo do ciclo evolutivo. Dado um conjunto de dados $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, uma rede neural N é uma sequência finita $((\boldsymbol{W}_1, \boldsymbol{b}_1, f_1), \ldots, (\boldsymbol{W}_n, \boldsymbol{b}_n, f_n))$ tal que a sinal o_N de N associada ao conjunto $\boldsymbol{X}$ é definido como:

$$o_N(\boldsymbol{X}) = f_n(f_{n-1}(f_{n-2}(\ldots)\boldsymbol{W}_{n-1} + \mathbf{b}_{n-1})\boldsymbol{W}_n + \mathbf{b}_n)$$

Onde $\boldsymbol{W}_i$ é uma matriz de pesos associada às conexões da camada $i-1$-ésima para a i-ésima camada, b_i um vetor denominado *biases* e f_i uma função denominada função de ativação da camada i.

a) Funcões de ativação: são funções associadas à cada camada de uma rede neural. São fundamentais para limitar o sinal produzido por uma unidade a um intervalo desejado, bem como inserir não-linearidade neste mesmo sinal.

Duas das muitas funções de ativações utilizadas nos experimentos são a $\tanh$ e $softmax$.

- $\tanh$ é útil a fim de escalar o somatório $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}_i + \mathbf{b}_i$ e contê-lo no intervalo $[-1, 1]$. $\tanh$ é definida como:

$$o(\boldsymbol{Z}) = \frac{e^{\boldsymbol{Z}} - e^{-\boldsymbol{Z}}}{e^{\boldsymbol{Z}} + e^{-\boldsymbol{Z}}}$$

- $softmax$ é frequentemente utilizada em tarefas de classificação. Dado c classes, $softmax$ tem exatas c unidades u_i (i.e., tem seus pesos descritos por uma matriz $[W_n]_{m \times c}$) onde $\sum_{i=1}^{c} o(u_i) = 1$. A saída de $o(u_i)$ pode portanto ser treinada para representar a probabilidade de uma amostra do conjunto de dados pertencer a classe i. $softmax$ é descrita abaixo:

$$o(\boldsymbol{X})_i = \frac{\boldsymbol{X}\boldsymbol{W}_i + \mathbf{b}}{\sum_{k=1}^{c} \boldsymbol{X}\boldsymbol{W}_k + \mathbf{b}}$$

b) Treinamento: utilizando o algoritmo de *gradient descent*, o vetor de sentido oposto ao gradiente de uma função de error pré-definida entre o sinal desejado e a saída de uma camada é adicionado aos parâmetros desta camada. Os sinais são então retro-propagados às camadas anteriores, a fim de que essas também se atualizem.

B. Algoritmos genéticos

Algoritmos genéticos tentam simular o processo de seleção natural e evolução descrito pela teoria Darwinista. Embora o conceito de "algoritmo genético" não seja fortemente definido, é possível observar ao menos quatro elementos em comum entre os métodos denominados "algoritmos genéticos" [4]: populações de indivíduos que podem ser descritos como cromossomos, método de seleção natural dos indivíduos por aptidão, um operador de *cross-over*, capaz de gerar filhos a partir dos indivíduos pertencentes à população e um operador de mutação, capaz de alterar localmente prontamente gerados.

Intuitivamente, este modelo prevê a passagem da população pelas fases de reprodução, mutação e seleção múltiplas vezes, prezando pela sobrevivência dos indivíduos mais "aptos". A aptidão de um indivíduo, por sua vez, é computada a partir de uma função definida de acordo com o domínio do problema, denominada *fitness*.

O procedimento 1 descreve um processo evolutivo genérico.

```
funcao algoritmo_genetico():
  populacao = gerar_populacao_inicial()

  enquanto deve_continuar_evoluindo():
    mais_apto = argmax(populacao, func=fitness)
    pares = selecionar_pares(populacao)

    filhos = [mutar(cross_over(A, B))
              para (A, B) em pares]

    populacao = selecionar(populacao, filhos)

  retorne mais_apto
```

Código 1. Descrição abstrada de um algoritmo genético.

Finalmente, a implementação das funções **gerar_populacao_inicial**, **deve_continuar_evoluindo**, **selecionar_pares**, **cross_over**, **mutar** e **selecionar** adapta o algoritmo genético base ao domínio do problema que está sendo tratado.

III. Trabalho proposto

Redes neurais artificias podem ser representadas de uma maneira simples: sequências finitas de triplas de matrizes, vetores e funções de ativação. Sabe-se, ainda, que o treinamento de uma rede neural pode ser traduzido como um problema de otimização. É possível, portanto, trivialmente visualizar uma rede neural como um indivíduo candidato à solução (a minimização de L). Desta forma, este trabalho propõe o treinamento de redes neurais utilizando um processo evolucionário descrito por um algoritmo genético.

A. Modelagem do problema

Primeiramente, a fim de simplificar o problema, restringe-se a estrutura de indivíduos pertencentes ao processo evolutivo: seja $\boldsymbol{X} \in \mathbb{R}^{n \times f}$ um conjunto de dados com n amostras e f características, $\boldsymbol{y} \in \mathbb{N}^n$ um vetor contendo, para cada amostra em $\boldsymbol{X}$, uma classe associada dentre as c possíveis e $N = ((\boldsymbol{W}_1, \boldsymbol{b}_1, f_1), (\boldsymbol{W}_2, \boldsymbol{b}_2, f_2))$, onde $\boldsymbol{W}_1 \in \mathbb{R}^{f \times g}$ e $\boldsymbol{W}_2 \in \mathbb{R}^{g \times c}$, $g \in \mathbb{N}$, com funções de ativação $tanh$ e $softmax$ na primeira e segunda camada, respectivamente. Define-se como um indivíduo do processo evolutivo uma rede neural pela qual pode-se alcançar N através de operações de soma de matrizes ou multiplicação por escalar às matrizes nas tuplas de N, bem como a soma de vetores ou multiplicação por escalar aos vetores nestas mesmas tuplas.

B. Geração Populacional Aleatória

A criação da população inicial se dá pela geração de p indivíduos, onde cada indivíduo tem cada uma de suas matrizes iniciadas com valores randômicos seguindo uma distribuição gaussiana com desvio padrão $\sqrt{\frac{1}{n}}$, onde n é o número de neurônios de entrada. Ademais, cada vetor de *bias* é iniciado com zeros.

C. Aptidão dos indivíduos da população

Seja $\boldsymbol{X}_{train} \in \mathbb{R}^{n \times f}$ um conjunto de testes e $\boldsymbol{y}_{train}$ um vetor de classes. $I = ((\boldsymbol{W}_1, \boldsymbol{b}_1, f_1), (\boldsymbol{W}_2, \boldsymbol{b}_2, f_2))$ um indivíduo do processo genético e $o_I = softmax(\tanh(\boldsymbol{X}\boldsymbol{W}_1 +$

$\boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2)$. A aptidão de I é definida por:

$$fitness(I) = -\sum_{x \in \boldsymbol{X}_{train}} (\arg\max(o_I) - y_i)^2$$

Em outras palavras, buscamos minimizar o número de amostras classificadas incorretamente.

D. Seleção para reprodução

A cada geração do processo evolutivo, indivíduos devem ser selecionados para compor pares aos quais o operador de *cross-over* será aplicado. Considerando que n indivíduos serão selecionados, esse processo pode ocorrer das seguintes maneiras:

- *roulette*: primeiramente, a *fitness* de cada indivíduo na população P é computada. Se o problema admite *fitness* negativas, é subtraído de todos os valores o menor valor observado no conjunto, transladando as *fitness* de todos para o intervalo positivo. Finalmente, n indivíduos são aleatoriamente selecionados, onde a probabilidade de que o indivíduo $I \in P$ seja selecionado é de $\frac{fitness(I)}{\sum_{F \in P} fitness(F)}$.

- *tournament*: Considerando que n indivíduos serão selecionados, exatos n "torneios" são realizados. Para cada torneio, uma parcela p da população é aleatoriamente selecionada e, destes, o indivíduo de maior *fitness* é escolhido. Nota-se que, para $p = |P|$, um mesmo indivíduo (com *fitness* máxima) é selecionado n vezes; enquanto, para $p = 1$, temos uma seleção completamente aleatória.

A seleção para reprodução é descrita pelo código 2

```
funcao selecionar_pares(populacao):
  finesses = [fitness(i) para i em populacao]

  se método-seleção-reprodução é roulette:
    fitness_min = min(finesses)
    se fitness_min < 0:
      p -= fitness_min
    fitness /= sum(fitness)

    retorne rand_escolhas(populacao,
      qtd=n, prob=fitness)

  se método-seleção-reprodução é tournament:
    retorne [
      argmax(
        rand_escolhas(população,qtd=tam_torneio),
        f=fitness)
      para i em (0, n)]
```

Código 2. O procedimento de seleção para reprodução

E. Operador de cross-over

O operador de *cross-over*, definido entre dois indivíduos, ocorre através da combinação entre suas matrizes e vetores de *bias*. Sejam A, B indivíduos do processo evolutivo, o *cross-over* entre eles é descrito pelo procedimento 3.

```
funcao cross_over(A, B):
    C = ()

    para i em (0, 1):
        W_a, b_a = A[i]
        W_b, b_b = B[i]
        (linhas, colunas) = W_a.forma

        ponto_corte = rand_int(0, linhas * colunas)

        # Percebe matrizes como vetores.
        W_a, W_b = W_a.plano(), W_b.plano()

        # Reconstroi uma matriz a partir dos vetores
        W_c = concat(W_a[:ponto_corte],
                W_b[ponto_corte:])
          .como_matriz((linhas, colunas))

        ponto_corte = rand_int(0, colunas)
        b_c = concat(b_a[:ponto_corte], b_b[
ponto_corte:])

        C.adicione((W_c, b_c))

    retorne C
```

Código 3. Operador de *cross-over* entre dois indivíduos.

Exemplifica-se agora, graficamente, a operação de *cross-over* para algum ponto de corte qualquer. Se A e B são descritos pela figura 2, temos um possível indivíduo resultante C como o ilustrado na figura 3.

Figura 2. Representação visual de duas redes neurais (ou indivíduos) A e B.

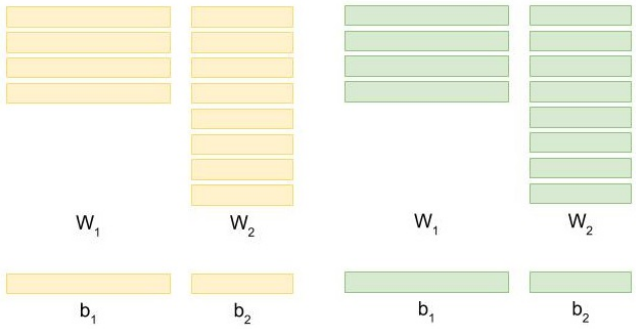
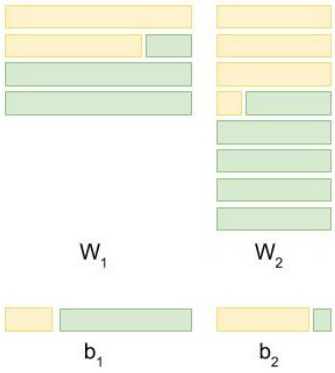


Figura 3. Ilustração do operador de cross-over aplicado à A e B, resultando na rede neural $\boldsymbol{C}$.



F. Operador de mutação

A mutação é definida sobre cada elemento real na matrizes de pesos e vetores de *biases*, para cada camada de uma

rede. Esta não ocorre pela troca de um valor por outro randômico, mas sim pela soma de um real contido no intervalo $[-fator, fator]$ ao valor anterior. Desta forma, a soma de elementos à já bons valores não os afeta abruptamente, enquanto valores ruins são atualizado gradualmente.

```
funcao mutar(C, fator, p):
  para i, (W, b) em enumerar(unir(C.Ws, C.bs)):
    # Define quais valores sofrerão mutação.
    mutantes = rand_floats(W.forma) < p
    # Muta valores.
    W[mutantes] += (2 * rand_int() -1) * fator

    mutantes = rand_floats(b.forma) < p
    b[mutantes] += (2 * rand_int() -1) * fator
  retorne C
```

Código 4. Procedimento para mutação genética de um indivíduo.

G. Seleção

Sejam $P_i = \{I_1, I_2, \ldots, I_n\}$ a geração de indivíduos i e $O_i = \{O_1, O_2, \ldots, O_m\}$ os filhos gerados pelo operador de *cross-over* sobre P_i. A seleção natural dos indivíduos em $P_i + O_i$ pode ocorre de duas maneiras distintas:

- *steady-state*: a nova população P_{i+1} é definida como $P_i^+ + O_i$, onde P_i^+ é o conjunto dos $n - m$ indivíduos em P_i com melhor valor de *fitness*.
- elistimo: $P_{i+1} = O_i + I^\star$, onde $I^\star$ é o indivíduo em P_i com melhor valor de *fitness* associado.

Algoritmicamente, este processo é descrito em 5.

```
funcao selecionar(populacao, filhos):
  populacao.ordenar(chave=(i): -fitness(i))
  filhos.ordenar(chave=(i): -fitness(i))

  se metodo é elitismo:
    populacao = concat(
      populacao[:tam(filhos)],
      filhos)
  se metodo é steady-state:
    populacao = concat(
      [populacao[0]],
      filhos)
  retorne populacao
```

Código 5. Procedimento para a seleção natural de indivíduos.

IV. Materiais e Métodos

Para validar os resultados obtidos, dois conjuntos de dados foram utilizados: o Iris Flower e o Spiral. Cada conjunto passou pelo processo de *standardizing* e foi separado em dois subconjuntos: D_{train} e D_{test}. Os testes ocorreram então em duas fases.

Na primeira fase, múltiplas RNs foram evoluídas pelo algoritmo genético descrito acima, utilizando diferentes parâmetros para a execução do mesmo. Os resultados foram então comparados em termos de: tempo de execução, utilidade por tempo, pontuação de acurácia do melhor indivíduo - definido como a porcentagem de acertos na classificação de amostras em D_{test}.

Na segunda fase, uma RN é treinada sobre D_{train} utilizando *gradient descent* e tem uma pontuação - definida como na fase 1 - computada sobre o conjunto D_{test}. Em seguida, uma segunda rede é treinada utilizando o processo evolutivo descrito acima - e os parâmetros com melhor resultado na primeira fase - e sua pontuação é computada sobre D_{test}. Informações empíricas são coletadas durante todo o procedimento.

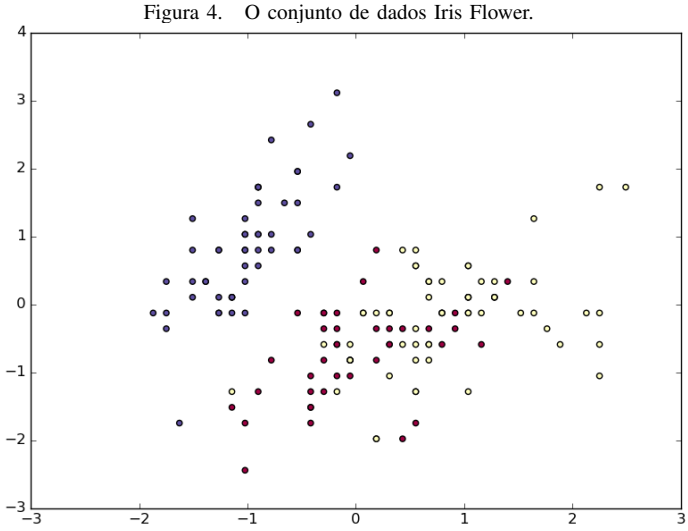
Vale destacar que os experimentos descritos aqui foram conduzidos em um micro computador Intel i7-4700 2.40GHz, 16 GB RAM em um ambiente GNU Linux Ubuntu 14.04.

V. Resultados e Discussão

A. Iris Flower

Iris Flower (figura 4) é um conjunto padrão na execução de testes simples em tarefas de classificação. Contendo 150 amostras com 4 características, cada amostra é associada a uma das três diferentes classes.

Figura 4. O conjunto de dados Iris Flower.



a) Fase 1: primeiramente, os seguintes parâmetros são considerados o padrão para a inicialização do algoritmo genético (tabela I).

tamanho da população (TP)	1000
máximo de ciclos evolutivos (MCE)	200
máxima duração (MD)	300 s
similaridade genética mínima (SGM)	0
método de reprodução (MR)	roulette
tamanho do torneio (TT)	-
# selecionados para reprodução (SR)	1.0
fator de mutação (FM)	.2
probabilidade de mutação (PM)	.1
método de seleção natural (MSN)	steady-state

Tabela I
Parâmetros do algoritmo genético e seus valores padrões.

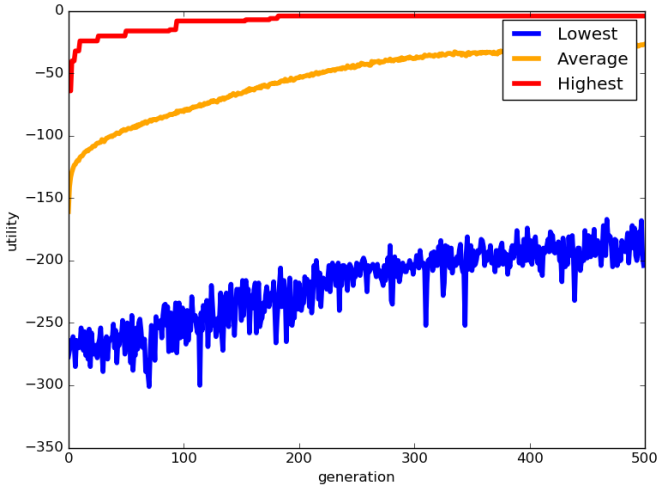
A tabela II descreve como as variações de parâmetros afetaram as variações de tempo, ciclos evolutivos e pontuação. A primeira vista, os resultados foram fracamente influenciados pela mudança dos parâmetros, mas a análise do histórico de *fitness* (figura 5) revela como o sexto AG otimizou perfeitamente a *fitness* da RN ao conjunto de treino, embora tenha apresentado uma pior pontuação em relação ao conjunto

de testes. Este evento claramente caracteriza-se como super-especificação ou *overfitting*.

#	Parâmetros	Duração	# ciclos	Acurácia
1	*padrão*	26.28 s	200	.90
2	TP: 100	**2.57 s**	200	**.93**
3	TP: 10000	263.59 s	200	.90
4	MCE: 0	.21 s	0	.80
5	MD: 10 s	10.06 s	77	.87
6	MD: 600 s, MCE: 500	68.91 s	500	.90
7	MR: *tournament*, TT: .1	300.28 s	152	.90
8	MR: *tournament*, TT: .8	301.73 s	132	**.93**
9	MSN: elitismo	54.64 s	200	**.93**
10	SR = .5	13.94 s	200	**.93**
11	SR = .25	7.42 s	200	.90
12	PM = .6	27.82 s	200	.9

Tabela II
VARIAÇÃO DE RESULTADOS A PARTIR DE MUDANÇAS NOS PARÂMETROS DO ALGORITMO GENÉTICO.

Figura 5. As aptidões dos indivíduos gerados durante os ciclos evolutivos no AG 6.



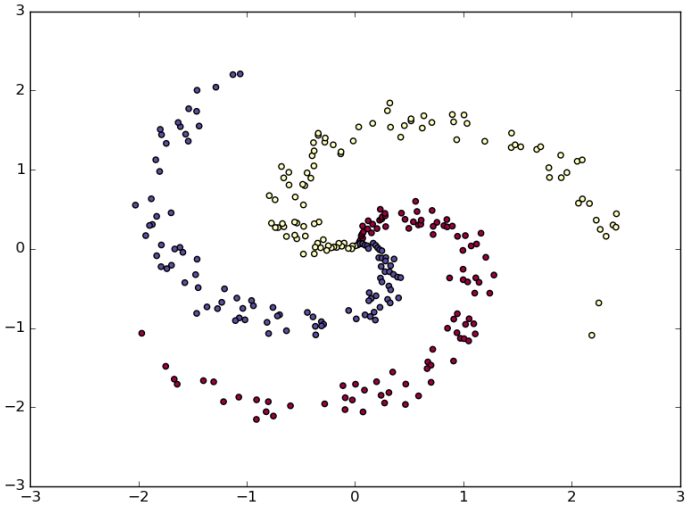
Por fim, nota-se pela figura 5 como o aumento da *fitness* média na população ocorre de forma logaritmica. Este comportamento foi também observado em [2], onde o autor o atribui ao caráter global de AG: ao explorar o espaço de busca, ele facilmente encontra boas soluções "genéricas", mas posteriormente apresenta dificuldades em melhorar a solução, visto que este "fine-tuning" se baseia fortemente no operador mutação.

b) Fase 2: para a segunda fase, uma RN foi treinada com *gradient descent*, precisando de 0.03 s para completar o treinamento, esta apresentou acurácia de 0.97 sobre o conjunto de testes. Claramente, esta apresenta resultados muito mais atraentes que qualquer modelo evoluído na fase 1.

B. Spiral

O conjunto de dados **Spiral**, contendo 300 amostras separadas por três classes, é ilustrado na figura 6.

Figura 6. O conjunto de dados Spiral.



a) Fase 1: os mesmo parâmetros exibidos em **??** são considerados o padrão para a inicialização do algoritmo genético aqui.

A tabela III descreve como as variações de parâmetros afetaram as variações de tempo, ciclos evolutivos e pontuação. Desta vez, o sétimo AG apresentou a melhor acurácia (97%), embora tenha sido exigido exaustivos 219.65 segundos para sua conclusão.

#	Parâmetros	Duração	# ciclos	Acurácia
1	*padrão*	-	-	-
2	TP: 100	4.46 s	200	.85
3	TP: 10000	301.44 s	135	.68
4	MCE: 0	**.39 s**	0	.43
5	MD: 10 s	10.09 s	45	.65
6	MD: 600 s, MCE: 500	108.17 s	500	.80
7	MD: 600, MCE: 1000	219.65 s	1000	**.98**
8	MR: *tournament*, TT: .1	300.63 s	168	.93
9	MR: *tournament*, TT: .8	301.62 s	141	.95
10	MSN: elitismo	82.66 s	200	.78
11	SR = .5	20.94 s	200	.77
12	SR = .25	10.95 s	200	.60
13	PM: .6	41.82 s	200	.68

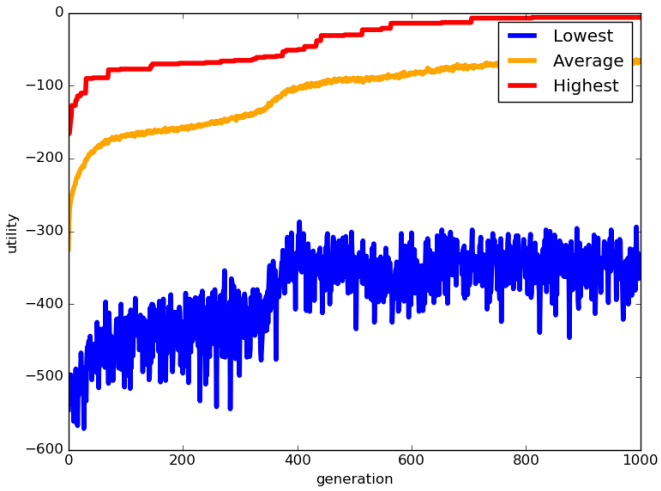
Tabela III
VARIAÇÃO DE RESULTADOS A PARTIR DE MUDANÇAS NOS PARÂMETROS DO ALGORITMO GENÉTICO.

Nota-se, pela figura 7, que AG 8 alcançou um indivíduo com valor de *fitness* ótimo, exatamente como AG 6 na Iris Flower. Entretanto, aqui observamos um alto valor de acurácia de predições sobre o conjunto de teste, o que sugere que o modelo generalizou corretamente os dados contidos em treino e teste.

b) Fase 2: treinada com *gradient descent*, uma RN obteve pontuação 1 no conjunto de testes exigindo somente .11 segundos para o treino. Isto é, esta apresentou uma melhor performance que a melhor RN evoluída na fase 1, embora ambas apresentem acurácias similares.

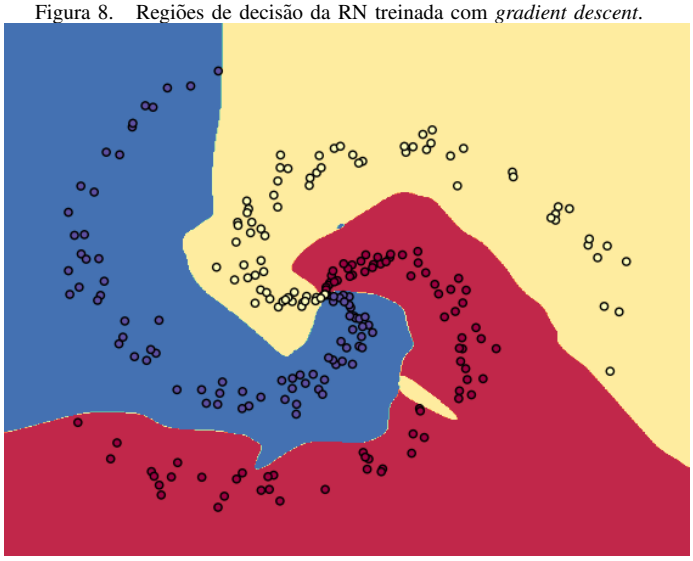
Como Spiral tem exatas duas dimensões, pode-se criar um

Figura 7. As aptidões dos indivíduos gerados durante os ciclos evolutivos no AG 7.



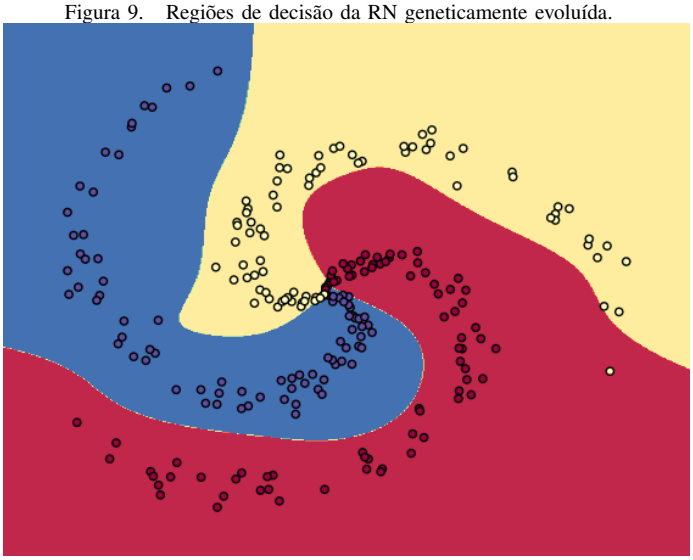
mesh de amostras e entregá-la a uma RN, que retornará as classes associadas. É possível, então, colorir as regiões de decisões das redes no gráfico sem grandes confusões, o que ocorreria para conjuntos de maior dimensionalidade. Figuras 8 e 9 ilustram as fronteiras de decisão das redes neurais consideradas na fase 2.

Figura 8. Regiões de decisão da RN treinada com *gradient descent*.



Enquanto as fronteiras criadas pela rede treinada com *gradient descent* apresentam curvas relativamente abruptas, as da rede evoluída se apresentam como curvas suaves. De fato, todas as execuções do algoritmo genético apresentaram esta característica em seus grafos de fronteiras de decisão. Especula-se que isso ocorra pela baixo "fine-tuning"nos modelos evoluídos.

Figura 9. Regiões de decisão da RN geneticamente evoluída.



VI. Conclusões

Neste trabalho, o treinamento de um conjunto restrito de redes neurais artificiais foi explorado empiricamente sobre dois conjuntos de dados. Observou-se nos experimentos que a acurácia de ambos os métodos é comparável, embora *gradient descent* sempre apresentasse uma solução com maior performance em tempo e espaço. Como apresentado por Koehn, foi verificado o crescimento logarítimico de *fitness* na evolução de RN, além de fronteiras de decisões suaves nos modelos evoluídos terem sido observados. Embora a brevidade dos experimentos e limitação em relação à simplicidade da RN e dos conjuntos de dados utilizados, a modelagem do treinamento e ambiente criados poderiam ser trivialmente reutilizados para o treinamento sobre conjuntos de dados maiores e redes com maior número de camadas e unidades. Como trabalhos futuros possíveis, seria interessante explorar uma integração do treinamento por evolução à bibliotecas de aprendizado de máquina bem conceituadas, que apresentam implementações eficientes de redes neurais; a extensão deste treinamento para outras redes que apresentem topologias variáveis, redes convolucionais e recorrentes; bem como a aplicação do treinamento por evolução sobre conjuntos de dados mais complexos.

Referências

[1] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms." in *IJCAI*, vol. 89, 1989, pp. 762–767. 1

[2] P. Koehn, "Combining genetic algorithms and neural networks: The encoding problem," 1994. 1, 5

[3] B. YEGNANARAYANA, *ARTIFICIAL NEURAL NETWORKS*. PHI Learning, 2009. 1

[4] M. Mitchell, *An Introduction to Genetic Algorithms*, ser. A Bradford book. Bradford Books, 1998. 2