Universidade Federal de Uberlândia

Faculdade de Engenharia Elétrica

3º trabalho de Inteligência Artificial

Regra de Hebb para treinamento de redes neurais artificiais

Aluno
Roní G. Gonçalves
10921EEL026

Professor
Keiji Yamanaka

Uberlândia, 5 de maio de 2014

Resumo

Um programa usando o método de treinamento de neurônios artificias pela regra de Hebb é desenvolvido para implementar todas as portas lógicas com duas entradas. O programa foi criado com a combinação de Python, GTK+ 3, GtkBuilder e Glade.

Palavras-chave: inteligência artificial, neurônios artificiais, regra de Hebb, portas lógicas, Python, GTK+ 3.

Abstract

An application that implements all the two-entries logic gates was developped using the Hebb's rule to train the artificial neural network. It was created using Python, GTK+ 3, GtkBuilder and Glade.

Keywords: artificial intelligence, artificial neurons, Hebb's rule, logic gates, Python, GTK+ 3.

Sumário

1 Introdução

Por meio da combinação da linguagem de programação Python, do editor gráfico de interfaces Glade, da biblioteca gráfica GTK+ 3 e do GtkBuilder[1] desenvolvi um programa com interface gráfica que ilustra o uso da regra de Hebb para treinar uma rede neural artificial, que é capaz de recriar o comportamento de qualquer porta lógica de duas entradas.

Primeiramente, apresento a regra de Hebb para o treinamento da rede neural artificial. Logo após, cada elemento necessário à criação do programa é brevemente explicado.

Ao final, algumas conclusões são tomadas a partir dos resultados. Algumas perspectivas de melhorias no programa são apresentadas.

Todos os códigos usados no programa se encontram presentes no anexo deste trabalho.

2 A regra de Hebb

A regra de Hebb[2] foi uma das primeiras propostas de treinamento de redes neurais artificiais.

Tal regra pode ser enunciada da seguinte forma: *se dois neurônios interligados estiverem ativados ou desativados simultaneamente, o peso desta conexão deve ser aumentado*.

2.1 Algoritmo para duas entradas e uma saída

Considerando que as entradas e as saídas só possam assumir dois valores, o seguinte algoritmo se aplica para a determinação dos pesos das sinapses dos neurônios artificiais:

1. Inicializar os pesos iguais a zero: $w_1 \leftarrow 0$; $w_2 = 0$ e $b \leftarrow 0$.
2. Para cada par entradas-saída ($[e_1, e_2]$, t), fazer: $x_1 \leftarrow e_1$; $x_2 \leftarrow e_2$; $y \leftarrow t$.
3. Definir: $\Delta w_1 \leftarrow x_1 y$, $\Delta w_2 \leftarrow x_2 y$.
4. Atualizar os pesos: $w_1 \leftarrow w_1 + \Delta w_1$, $w_2 \leftarrow w_2 + \Delta w_2$ e $b \leftarrow b + y$.

[1] Módulo responsável por fazer a ligação entre a interface gráfica feita no Glade em formato XML e o código desenvolvido em Python.

[2] Também conhecida como teoria hebbiana, nome em homenagem ao pesquisador Donald Hebb.

3 Programação

O programa *gamb.IA.rra* foi escrito em Python (www.python.org), que segundo o próprio site: *é uma linguagem interpretada, interativa, orientada a objetos. Ela incorpora módulos, exceções, tipagem dinâmica, estruturas de dados altamente dinâmicas e classes. O Python combina grande poder com sintaxe clara. Ele possui interfaces para inúmeras chamadas de sistemas e bibliotecas, assim como para inúmeros sistemas operacionais e ainda é extensível em C e C++* [...]

3.1 Esboço da interface gráfica

Na figura 2 é mostrada como a janela principal do programa deveria ser. O esboço foi feito no programa Pencil (http://pencil.evolus.vn/)

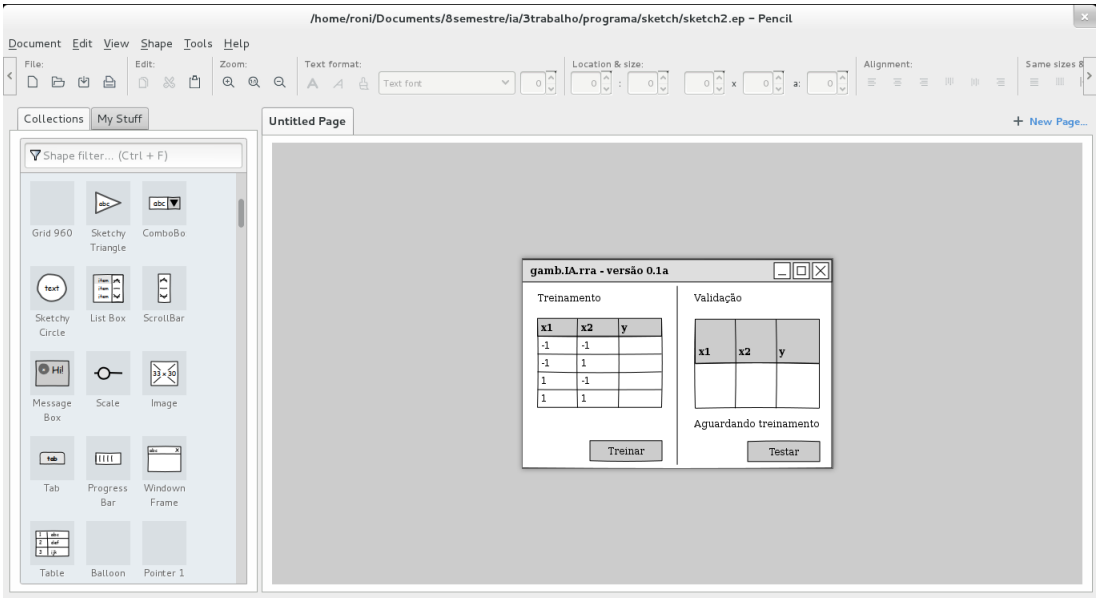

Figura 1: Janela principal do programa Pencil.

No lado esquerdo da janela principal, existe a área *Treinamento* onde o usuário pode inserir os valores da saída que ele quiser: isso caracteriza uma porta lógica. Variando os valores de saída, o usuário pode ter portas lógicas *e*, *ou*, *não e* ...

Após inserir os quatro valores desejados, o usuário deve pressionar o botão *Treinar*, dessa forma o algoritmo apresentado na seção 2.1 é usado

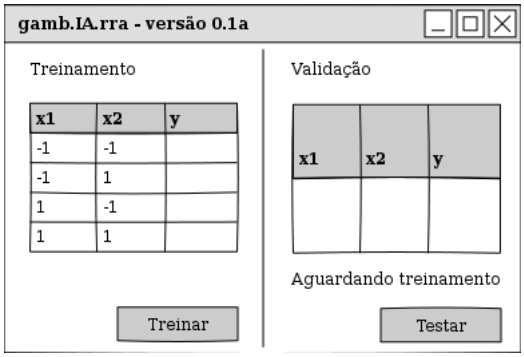

Figura 2: Esboço da interface gráfica do programa gamb.IA.rra.

e os pesos são determinados. Quando o treinamento termina, o rótulo *Aguardando treinamento* presente na área *Validação* à esquerda da janela principal indica que está pronta passando a exibir o texto *Treinamento completo*.

Uma vez feito o treinamento, o usuário pode testar se os pesos calculados funcionam realmente para quaisquer entradas x_1 e x_2 que ele digite na área *Validação*.

3.2 Interface gráfica em XML

O programa Glade (glade.gnome.org) foi usado para desenhar a interface gráfica proposta na figura 2. A janela principal do Glade pode ser vista logo a seguir.

3.3 Construtor GTK+

O GtkBuilder é o responsável por pegar o arquivo XML criado no Glade e transformá-lo em objetos de interface gráfica do GTK+. Além disso, ele associa os eventos ocorridos ao usar o programa como, por exemplo, clicar um botão aos seus objetos denominados *handlers*.

```
#Construindo interface e associando sinais a callbacks
builder = Gtk.Builder()
builder.add_from_file('../ui/interface3.ui')
builder.connect_signals(handlers)
```

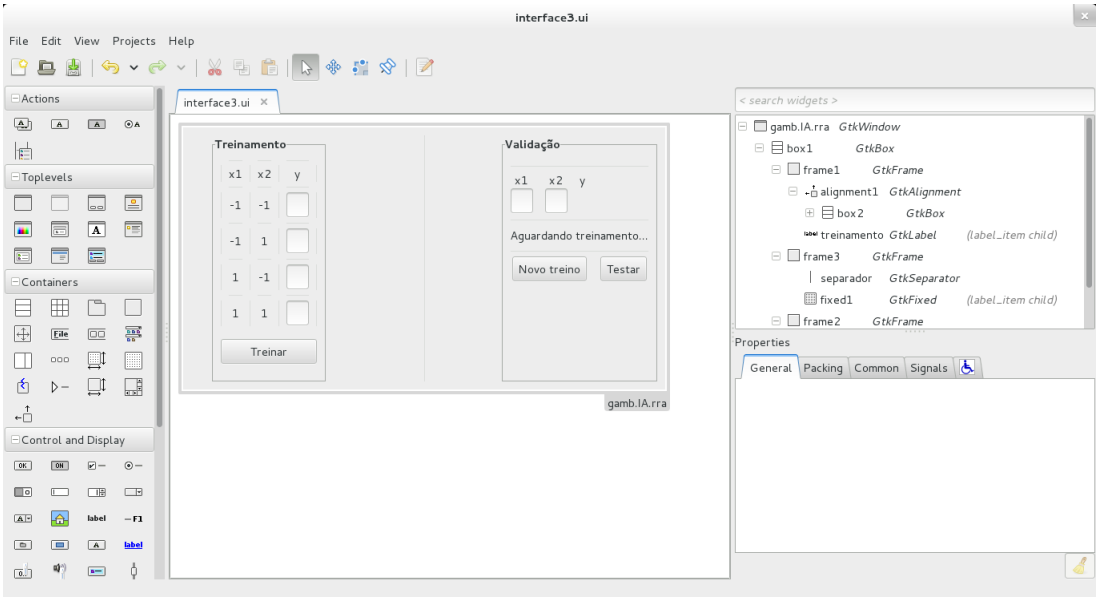

Figura 3: Janela principal do programa Glade usado para criar a interface gráfica do programa gamb.IA.rra.

3.4 Programa feito: gamb.IA.rra

Com o uso de todas as ferramentas citadas acima, o resultado final pode ser conferido na figura 4

Um caso de uso é mostrado na seqüência: o usuário digita os valores de saída que ele quer (figura 5), clica no botão *Treinar* (figura 6), testa se o programa aprendeu o comportamento da porta lógica com o par de entrada igual a 1 (figura 7) clicando no botão *Testar* (figura 8).

4 Conclusões

A regra de Hebb constitui-se numa primeira tentativa de determinar os pesos das ligações sinápticas de neurônios artificiais. Em vez de usar o método da tentativa e erro, pode-se aplicá-la com razoável sucesso para implementar portas lógicas.

Há um porém: somente problemas linearmente separáveis podem ser resolvidos com este método. De tal forma que os pesos obtidos para as portas lógicas *ou exclusivo* e *não ou exclusivo* não são válidos.

A interface gráfica criada ajuda no entendimento e no uso dos neurônios

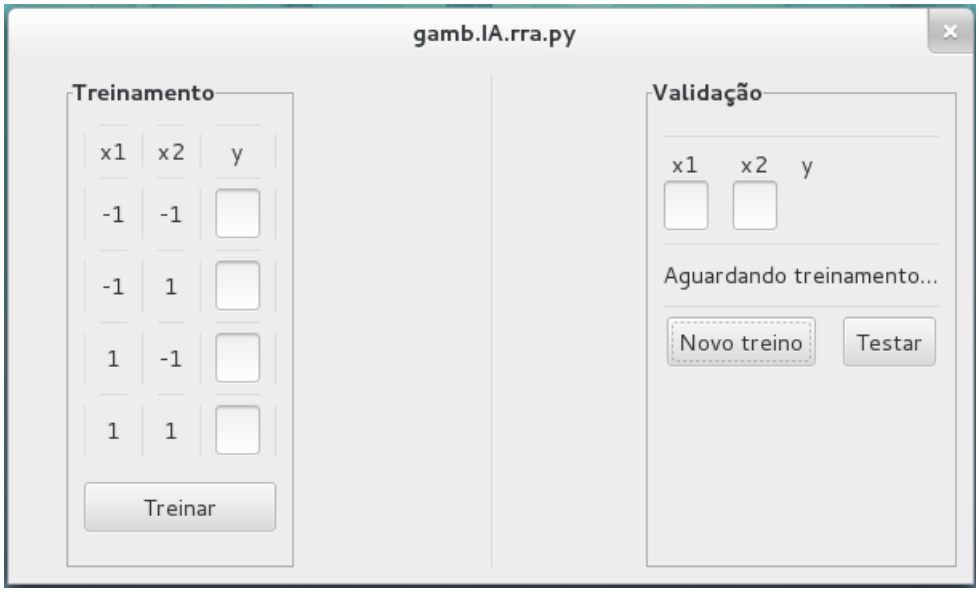

Figura 4: Janela principal e única do programa feito.

artificiais. No entanto, muitas melhorias ainda podem e serão feitas.

Figura 5: O usuário entra com os dados desejados para criar uma tabela-verdade e, conseqüentemente, uma porta lógica de duas entradas para realizar o treinamento da rede neural.

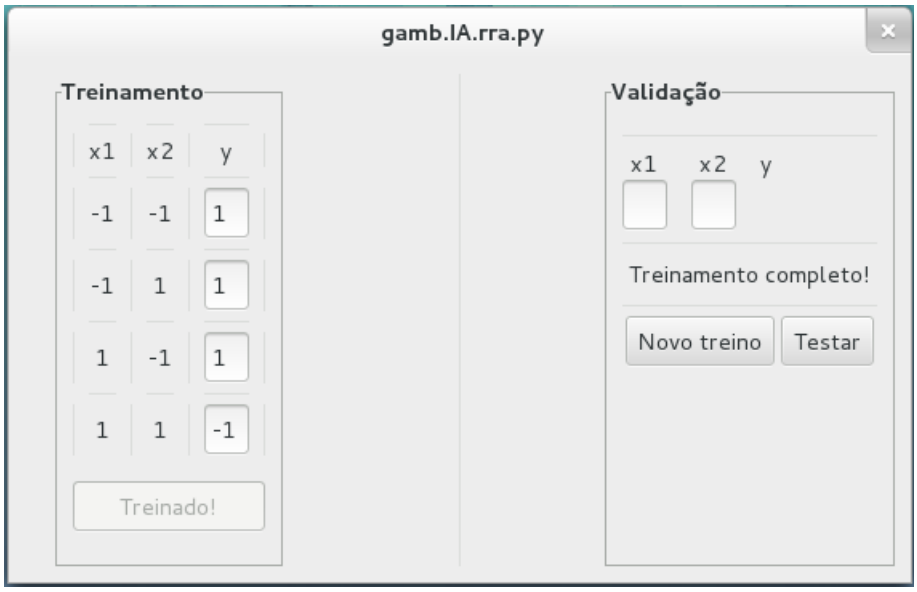

Figura 6: Após clicar no botão *Treinar*, os pesos são determinados pela regra de Hebb e o usuário pode, então, testar se a rede neural aprendeu corretamente a tabela verdade.

5 Referências

[1] Keiji Yamanaka, *Notas de aula da disciplina Inteligência Artificial*. Faculdade de Engenharia Elétria, Universidade Federal de Uberlândia, 2014.

[2] Kenneth Lambert, *Fundamentals of Python: From first programs through data structures*. Cengage Learning, 2010.

[3] Sebastian Pölsterl, *The Python GTK+ 3 tutorial*. Versão 3.4, 24 de abril de 2014.

Anexo

Listing 1: ../programa/code/gamb.IA.rra.py

```
#!/usr/bin/python

'''
Aluno: Roni Gilberto Goncalves
Matricula: 10921EEL026

Versao do programa: 0.1a

'''

from gi.repository import Gtk
from array import array

#Variaveis globais
w1 = 0
w2 = 0
b = 0
x1 = array('i', [-1, -1, 1, 1])
x2 = array('i', [-1, 1, -1, 1])
saida = array('i')

#Funcoes comuns
def func_soma(entrada1, entrada2, bias, peso1, peso2):

    return ((peso1*entrada1) + (peso2*entrada2) + bias)

def func_ativa(net):

    if net < 0:
        sinal = -1

    else:
        sinal = 1

    return sinal
```

```
#Funcoes de callback
def func_treinar(widget, *events):

    global saida
    global w1
    global w2
    global b

    Y1 = y1.get_text()
    Y2 = y2.get_text()
    Y3 = y3.get_text()
    Y4 = y4.get_text()

    saida.append(int(Y1))
    saida.append(int(Y2))
    saida.append(int(Y3))
    saida.append(int(Y4))

    for i in range(3):

        d1 = x1[i]*saida[i]
        d2 = x2[i]*saida[i]

        w1 = w1 + d1
        w2 = w2 + d2
        b = b + saida[i]

    d1 = x1[3]*saida[3]
    d2 = x2[3]*saida[3]

    w1 = w1 + d1
    w2 = w2 + d2
    b = b + saida[3]

    print w1
    print w2
    print b
    print d1
    print d2
```

```
    botaoTreinar.set_label('Treinado!')
    botaoTreinar.set_sensitive(False)
    LabelStatus.set_text('Treinamento completo!')

def func_testar(widget, *events):

    E1 = e1.get_text()
    E2 = e2.get_text()

    E1 = int(E1)
    E2 = int(E2)

    resposta = func_ativa(func_soma(E1, E2, b, w1, w2))
    resposta = str(resposta)

    LabelResultado.set_text(resposta)

def func_novoTreino(widget, *events):

    global w1
    global w2
    global b

    w1 = 0
    w2 = 0
    b = 0

    botaoTreinar.set_sensitive(True)
    botaoTreinar.set_label('Treinar')
    LabelResultado.set_text('')
    LabelStatus.set_text('Aguardando treinamento...')

    y1.set_text('')
    y2.set_text('')
    y3.set_text('')
    y4.set_text('')

    e1.set_text('')
    e2.set_text('')
```

```
#Dicionario de eventos
handlers = {
    'clicaTreinar': func_treinar,
    'clicaTestar' : func_testar,
    'clicanovoTreino' : func_novoTreino,
}

#Construindo interface e associando sinais a callbacks
builder = Gtk.Builder()
builder.add_from_file('../ui/interface3.ui')
builder.connect_signals(handlers)

#Configurando janela
win = builder.get_object('gamb.IA.rra')
win.connect('destroy', Gtk.main_quit)

#Obtendo widgets para posterior manipulacao

LabelStatus = builder.get_object('status')
LabelResultado = builder.get_object('saida_teste')

e1 = builder.get_object('entrada1')
e2 = builder.get_object('entrada2')

y1 = builder.get_object('saida1')
y2 = builder.get_object('saida2')
y3 = builder.get_object('saida3')
y4 = builder.get_object('saida4')

botaoTreinar = builder.get_object('b_treina')
botaoTestar = builder.get_object('b_testa')
botaonovoTeino = builder.get_object('b_novoTreino')

#Exibindo interface e iniciando loop de eventos
win.show_all()
Gtk.main()
```

Figura 7: O usuário entra com os dados para realizar o teste e conferir se o resultado é igual ao esperado.

Figura 8: O usuário confirma que para o par de entradas (1,1) gera a saída esperada -1.