Otimização do Erro de Treinamento de Redes Neurofuzzy

Vinicius Ferraz
Engenharia de Sistemas
Universidade Federal de Minas Gerais
Belo Horizonte
Email: vinicius.ferraz@tjmg.jus.br

***Abstract*—Single-objective optimization.**

I. Introdução

ANFIS é uma sigla para Sistema de Inferência Neurofuzzy Adaptativo.

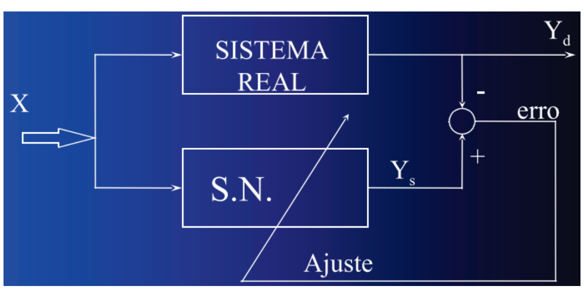
Considere um Sistema Real com entradas $x \in \mathbb{R}^V$ e saída $y \in \mathbb{R}$. Nosso Sistema Nebuloso vai ajustar seus parâmetros internos com $|t|$ pontos na fase de treinamento, para depois fazer previsões com $|v|$ pontos na fase de validação.

Queremos minimizar o erro de treinamento

$$e = 0.5 \sum_{k=1}^{|t|} (y_{sk} - y_{dk})^2 = f(c_{ij}, \sigma_{ij}, p_{ij}, q_j)$$

.

Como $1 \leq i \leq V =$ nVariáveis e $1 \leq j \leq G =$ nGaussianas, então a cardinalidade de parâmetros é $3GV + G$.



Cada x_i poderá pertencer a G conjuntos nebulosos. As funções de pertinência são gaussianas unidimensionais: $\mu_{Aij}(x_i) = \exp\left[-0.5\left(\frac{x_i - c_{ij}}{\sigma_{ij}}\right)\right]$.

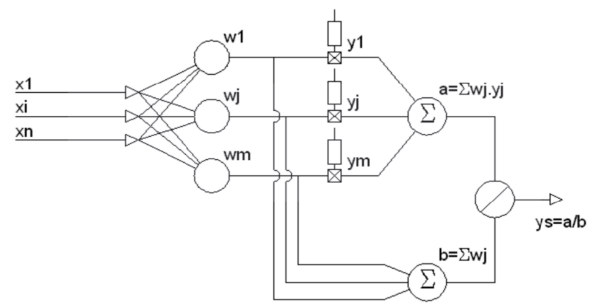
As regras nebulosas são: Se x_1 é A_{1j} e x_2 é A_{2j} $\cdots$ e x_n é A_{Vj}, então $y_j = \sum_{i=1}^{V} p_{ij} x_i + q_j$.

O precedente AND é calculado pela T-norma do produto algébrico: $w_j = \prod_{i=1}^{V} \mu_{Aij}(x_i)$.

A saída é calculada pela média ponderada:

$$y_s = \frac{\sum_{j=1}^{G} w_j y_j}{\sum_{j=1}^{G} w_j}$$

.



II. Cálculo do Gradiente

$\nabla e(c, \sigma, p, q)$ é calculado pela regra da cadeia:

$$e = \frac{1}{2}(y_s - y_d)^2 \Rightarrow \frac{\partial e}{\partial y_s} = \frac{1}{2} \cdot 2(y_s - y_d) = y_s - y_d$$

$$a = \sum_{j=1}^{m} w_j y_j$$

$$b = \sum_{j=1}^{m} w_j$$

$$y_s = \frac{a}{b} \Rightarrow \frac{\partial y_s}{\partial w_j} = \frac{y_j \cdot b - a \cdot 1}{b^2} = \frac{y_j - a/b}{b} = \frac{y_j - y_s}{b}$$

$$\frac{\partial y_s}{\partial y_j} = \frac{w_j}{b}$$

$$\mu_{Aij} = \exp\left[-\frac{1}{2}\left(\frac{x_i - c_{ij}}{\sigma_{ij}}\right)^2\right]$$

$$w_j = \prod_{i=1}^{n} \mu_{Aij} \Rightarrow \frac{\partial w_j}{\partial c_{ij}} = w_j \left(-\frac{1}{2}\right) \frac{\partial}{\partial c_{ij}} \left(\frac{x_i - c_{ij}}{\sigma_{ij}}\right)^2$$

$$= w_j \left(-\frac{1}{2}\right) \cdot 2 \left(\frac{x_i - c_{ij}}{\sigma_{ij}}\right) \left(-\frac{1}{\sigma_{ij}}\right)$$

$$= w_j \cdot \frac{x_i - c_{ij}}{\sigma_{ij}^2}$$

$$\frac{\partial w_j}{\partial \sigma_{ij}} = w_j \left(-\frac{1}{2}\right)(x_i - c_{ij})^2 \frac{\partial}{\partial \sigma_{ij}} \left(\sigma_{ij}^{-2}\right)$$
$$= w_j \cdot \frac{(x_i - c_{ij})^2}{\sigma_{ij}^3}$$
$$y_j = \sum_{i=1}^{n} p_{ij} x_i + q_j \Rightarrow \frac{\partial y_j}{\partial p_{ij}} = x_j$$
$$\frac{\partial y_j}{\partial q_j} = 1$$

III. Algoritmo Anterior — Gradient Descent

Tínhamos ao nosso dispor o seguinte algoritmo:

- Dados alfa, nEpocas, nPontosT, xt, ydt, nPontosV, xv, ydv, nVariaveis, nGaussianas.
- Inicializar objp, objq, objc, objsigma = INPUT
- (1) Embaralhar índices de treinamento.
- (2) Calcular Saída Error (xt, ydt): retorna objSaidaOmega, objSaidaz, objSaidays, obj.saida.denom
- (3) Calcular as derivadas do erro em relação a INPUT.
- (4) INPUT = INPUT $-$ alfa $\times$ gradiente; repetir a partir de (1) para todos os pontos de treinamento;
- Calcular Saída Error (xv, ydv).
- Finalmente, calcular o Erro Percentual Médio APE = $\frac{100\%}{|v|} \sum_{k=1}^{|v|} \frac{y_{sk} - y_{vk}}{y_{sk}}$

Queríamos trocar o método do gradiente $(3, 4)$ por métodos quase Newton, para minimizar Saída Error.

IV. Novo Algoritmo — Programação Quadrática Sequencial

Tentamos com DFP, BFGS, Huang e Biggs, cada um com seção áurea ou não, a conclusão foi que não tínhamos como generalizar um bom xmin e xmax para todos os casos.

Tentamos com a função *fmincon* do MatLab, a conclusão foi que um mínimo era encontrado, contudo não era aceitável em todos os casos.

O que funcionou foi o SQP modificado, ou seja, "Schittkowski's Sequential Quadratic Programming method".

Encapsulamos as variáveis em um vetor $(c, \sigma, p, q) \to x$ unidimensional, ao final fizemos o inverso: $x \to (c, \sigma, p, q)$.

Inicializamos com c, σ igualmente espaçados; p, q = números aleatórios entre -1 e 1. Também tentamos inicializar p, q com um ponto fixo, através do método dos mínimos quadrados, o resultado não foi aceitável.

As funções para otimização foram chamadas de testeError(x) e gradError(x), utilizamos variáveis globais (xt, ydt, nGaussianas, nVariaveis) para fazer os cálculos dentro delas, uma vez que eram parâmetros e não deviam ser considerados em x, que é repassado ao SQP.

Plotamos os gráficos de erro de treinamento ao longo das iterações: oscilaram muito, tanto no método do gradiente quanto no SQP.

Utilizamos em 4 problemas 11 gaussianas, no último, tivemos que alterar para 7.

Durante o algoritmo, algumas variáveis σ passaram para zero. Evitamos divisão por zero na gaussiana:

```
if sigma == 0 sigma = 1e-3; end
```

Tivemos que executar diversas vezes, em algumas o algoritmo lançava exceção de: "hessiana singular" (descobri que esse erro desaparece ao proibir o produtório dos ResultOmega(j) de ser igual a zero). Muitas vezes ou o trainError ou o APE não é aceitável.

Por outro lado, o método do gradiente, cada vez que é executado, fazemos isso de oito formas diferentes, com parâmetros booleanos $000, 001, \cdots, 111$.

Registramos os 3 melhores resultados, com a seguinte diretiva: Não importa se o trainError final é grande, importa o APE. O que acontece é que não temos como minimizar a validação.

A. *Configurações do SQP*

Como vemos abaixo, colocamos opções de busca irrestrita; alteramos o critério de parada; o máximo número de avaliações da função foi alterado para 1000; mais abaixo, ao depurar reparei que algumas condições booleanas estavam retornando vazio e eu tive de trocar algumas variáveis para zero, a fim de que a condição passasse a ser True ou False.

```
lower = upper = [];
opts(3) = 1e-4; Termination tolerance on F.
opts(6) = -1; Termination criteria.

If opts(6) = (-1), Schittkowski's criteria is used:
KTO = abs(s'*fp) + sum(abs(u.*gv)) <= opts(3)
SCV = sum(g(g>0)) <= sqrt(opts(3))

opts(14) = 1000;  Maximum number of function evaluations.
```

linha 392:

```
if length(udelta) == 0
    u = [];
else
    u = udelta(1:lenv);
end
```

linha 546:

```
if length(z0p1) == 0
    z0p1 = 0;
end
```

V. Testes Planejados

A seguir, testamos com cinco problemas clássicos.

A. *Problema 1 — Parábola*

$f(x) = x^2$.

Treinado com o intervalo $[-2, 2]$ dividido em 500 partes iguais.

Validado com o intervalo $[-2 + 2/500, 2 - 2/500]$ dividido em 500 partes iguais.

Gradiente:

$i = 1$; trainError = 670.873656; APE = 5.968505%;

$i = 4$; trainError $= 606.053391$; APE $= 29.396484\%$;

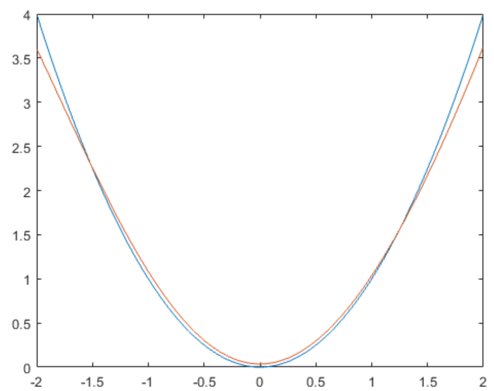
$i = 4$; trainError $= 598.511200$; APE $= 20.023147\%$;

SQP:

trainError $= 6.378561$; APE $= 19.043741\%$;

trainError $= 3.232768$; APE $= 16.698742\%$; gráfico abaixo;

trainError $= 1.429867$; APE $= 15.728088\%$;



B. *Problema 2 — Sistema Estático Multivariável — Seção 12.6.3 de [Jang]*

$f(x, y, z) = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2$.

Treinado com 216 linhas: de $[1, 1, 1]$ até $[6, 6, 6]$.

Validado com 125 linhas: de $[1.5, 1.5, 1.5]$ até $[5.5, 5.5, 5.5]$.

Gradiente:

$i = 2$; trainError $= 4148.678490$; APE $= 9.571080\%$;

$i = 8$; trainError $= 4088.343674$; APE $= 8.953101\%$;

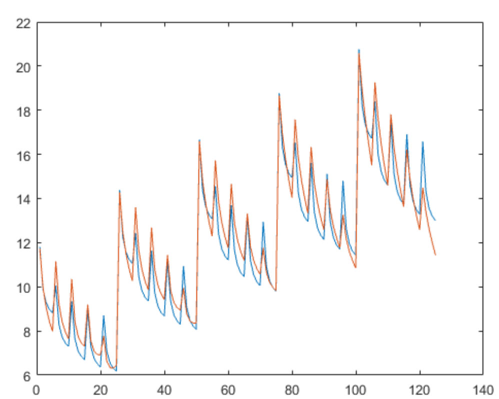
$i = 6$; trainError $= 4711.287320$; APE $= 5.646013\%$;

SQP:

trainError $= 503.0139745$; APE $= 7.377215\%$;

trainError $= 173.7602905$; APE $= 6.433070\%$;

trainError $= 385.418644$; APE $= 5.413850\%$; gráfico abaixo;



C. *Problema 3 — Sistema Dinâmico*

$y(k+1) = g(y(k), y(k-1), y(k-2), u(k), u(k-1))$;

$$g(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_3^2 + x_4^2};$$

u aleatório entre -1 e 1.

Foram geradas 5000 linhas aleatórias de treinamento e 5000 linhas aleatórias de validação.

Cada vez que eu rodava o script, eu conseguia valores ligeiramente diferentes.

Gradiente:

$i = 1$; trainError $= 207.581166$; APE $= 30.643149\%$;

$i = 2$; trainError $= 210.751646$; APE $= 28.685072\%$;

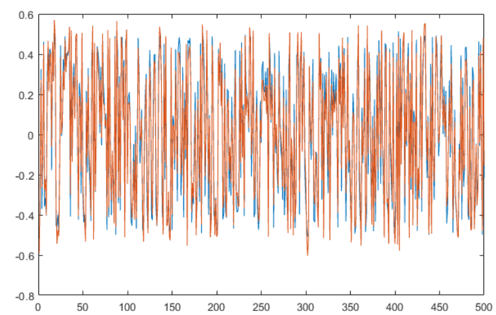
$i = 2$; trainError $= 205.709573$; APE $= 20.417532\%$;

SQP:

trainError $= 10.246691$; APE $= 90.890983\%$; gráfico abaixo;

trainError $= 11.6601635$; APE $= 55.929015\%$;

trainError $= 3.4975785$; APE $= 35.869743\%$;



D. *Problema 4 — Previsão de Séries Temporais — Seção 12.6.5 de [Jang]*

```
load mgdata.dat
xt(n,:) = [mgdata(t-18+1, 2)
           mgdata(t-12+1, 2)
           mgdata(t-6+1, 2)
           mgdata(t+1, 2)];
ydt(n) = mgdata(t+6+1, 2);
```

Ideia copiada de forma idêntica à de [Jang]:

Foram geradas 500 linhas para treinamento, sequencialmente a partir de $t = 118$.

Foram geradas 500 linhas de validação, sequencialmente a partir de $t = 118 + 500 = 618$.

Gradiente:

$i = 1$; trainError $= 25.297410$; APE $= 5.362822\%$;

$i = 3$; trainError $= 26.259203$; APE $= 3.898262\%$;

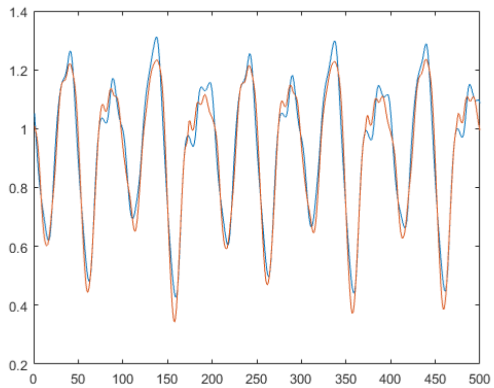
$i = 2$; trainError $= 24.992841$; APE $= 2.563632\%$;

SQP:

trainError $= 0.554332$; APE $= 4.854939\%$;

trainError $= 0.574025$; APE $= 4.806731\%$; gráfico abaixo;

trainError $= 0.3992405$; APE $= 3.932564\%$;



E. *Problema* 5 — *Regressão*

Link = https://archive.ics.uci.edu/ml/datasets/Energy+efficiency

Denominada Energy Efficiency, tem 8 colunas de entrada e 2 de saída, escolhi somente a nona para inicializar y_d.

A idéia é treinar com 10% dos dados e validar com 90%.

São 768 linhas. Treinei com as terminadas em 1:

$1, 11, 21, 31, \cdots, 761$.

Validei com as outras $768 - 77 = 691$ linhas.

Gradiente: (7 gaussianas)

$i = 6$; trainError = 122821.756013; APE = 36.741357%;

$i = 1$; trainError = 13147.204642; APE = 33.066422%;

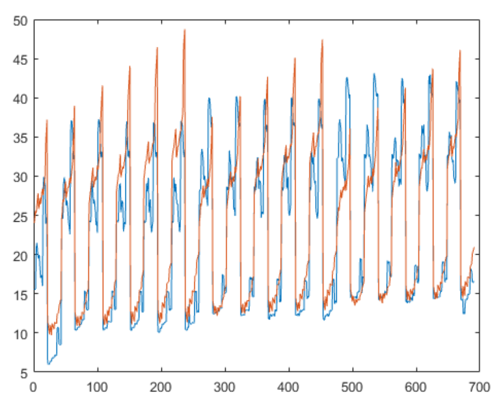
$i = 6$; trainError = 14146.733003; APE = 20.225644%;

SQP: (7 gaussianas)

trainError = 635.961791; APE = 13.918792%; gráfico abaixo;

trainError = 333.996546; APE = 11.321780%;

trainError = 302.098627; APE = 10.854442%;



VI. Conclusão

O erro de treinamento do SQP é muitas vezes inferior ao do método do gradiente:

Parábola: 598 versus 6.4; Razão = 93.4375;

Sistema Estático Multivariável: 4088 versus 503; Razão = 8.1272;

Sistema Dinâmico: 205.7 versus 11.7; Razão = 17.5812;

Previsão de Séries Temporais: 24.9 versus 0.58; Razão = 42.9310;

Regressão: 13147 versus 636; Razão = 20.6714;

Porém, precisamos melhorar a inicialização aleatória e a quantidade de vezes que o algoritmo é executado.

VII. Referência

Jang, Jyh-Shing Roger, Chuen-Tsai Sun, and Eiji Mizutani. "Neuro-fuzzy and soft computing-a computational approach to learning and machine intelligence [Book Review]."IEEE Transactions on automatic control 42.10 (1997): 1482-1484.