Universidade Federal de Minas Gerais

Nome: Guilherme Vinícius Amorim

Matrícula: 2017089081

Data: 10/2020

Exercício 10

O objetivo do exercício desta semana é observar que o MLP é capaz de aproximar qualquer função contínua. Para isso, foi realizada a regressão de um ciclo de uma senoide com backpropagation.

Inicialmente, os dados de treinamento e de testes foram criados, de acordo como uma senoide acrescida e um ruído. A Figura 1 mostra o código em R utilizado para criar esses dados.

```r
xtrain = seq(from = 0, to = 2 * pi, by = 0.15)
xtrain = xtrain + (runif(length(xtrain)) - 0.5)/5
ytrain = sin(xtrain)
ytrain = ytrain + (runif(length(ytrain)) - 0.5)/5
xtest = seq(from = 0, to = 2 * pi, by = 0.01)
ytest = sin(xtest)
```

Figura 1: Criação dos dados de treinamento e de teste.

A partir dos dados criados, uma RNA foi desenvolvida com três neurônios na camada escondida e um neurônio na camada de saída com função der ativação linear. Para o ajuste dos pesos z da matriz escondida, o método backpropagation foi utilizado. O código utilizado para o treinamento da rede MLP com o ajuste de peso com backpropagation está em anexo ao final deste relatório.

Como há um fator de aleatoriedade da eficácia do treinamento MLP, treinou-se 5 vezes os dados de treinamento. Para cada treinamento, calculou-se o erro médio quadrático percentual (MSE). A Figura 2 mostra o algoritmo utilizado nos 5 treinamentos. A Figura 3 mostra valor do MSE para cada rodada de treinamento, a média desses 5 valores de erro, e o desvio padrão.

Os parâmetros utilizados no treinamento foram:

$$eta = 0.01;\ tol = 0.1;\ maxepoch = 3000$$

```
MLP_vec<-c()

for (i in 1:5){

  # MLP Training
  MLP<-trainMLP(xtrain, ytrain, 3, 0.01, 0.1, 3000, 0)
  # ytest
  YTestHat<-yMLP(xtest, MLP, 0)
  # Calc percentage mean squared error (MSE)
  MSE<-(sum(YTestHat-ytest)^2)/length(YTestHat)

  MLP_vec<-c(MLP_vec, MSE)
}

mean_MLP <- mean(MLP_vec)
sd_MLP <-sd(MLP_vec)
```

Figura 2: Código utilizado no treinamento e cálculo de MSE.

```
> MLP_vec
[1] 0.02916709 0.17398106 0.06275701 0.19491544 0.05218049
> mean_MLP
[1] 0.1026002
> sd_MLP
[1] 0.07605817
```

Figura 3: O valor do MSE para cada rodada de treinamento, a média desses 5 valores de erro, e o desvio padrão.

Analisando os resultados, observa-se valores erro altos. No quarto treinamento, o erro percentual foi 20%. Além disso, observa-se um desvio padrão também alto, mostrando que o fator de aleatoriedade tem grande importância no treinamento. A Figura 4 mostra o *plot* das saídas. Os pontos em preto são os dados utilizados no treinamento. As funções em vermelho e azul são as utilizadas para treinamento e a saída treinada pela rede ELM, respectivamente.

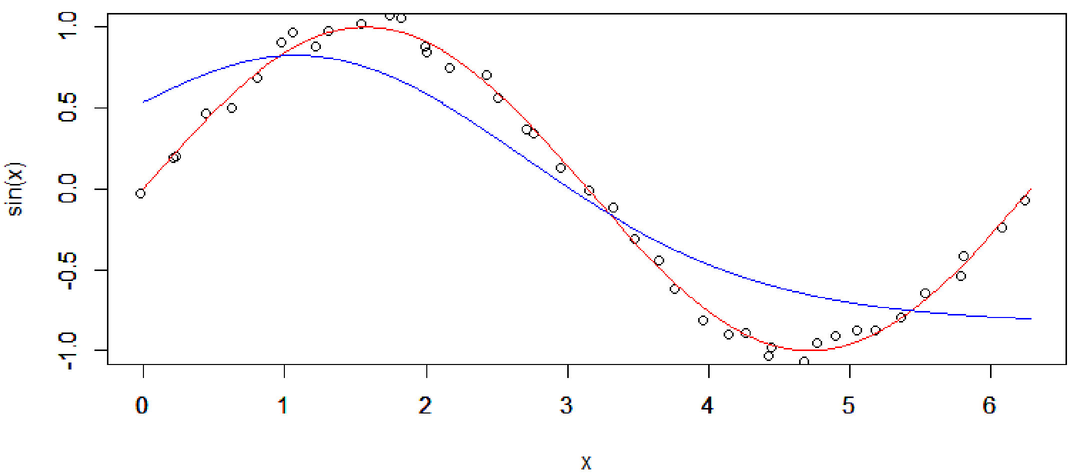

Figura 4: *Plot* das saídas senoidais de teste, treinamento e rede treinada.

Conclusão

O alto erro no treinamento pode ter ocorrido em função do baixo número de neurônios na camada escondida. Além disso, a tolerância utilizada foi 0,1. Uma tolerância menor pode reduzir o valor de erro. Ademais, o número de dados de treinamento pode ter sido não suficiente. Mais dados de treinamento pode melhorar o resultado esperado.

O algoritmo da rede de treinamento MLP pode ser visto na próxima página.

```
trainMLP <- function(X,Y,p,eta,tol,maxepocas,func_ati){

  xin<-as.matrix(X)
  yd<-as.matrix(Y)

  N<-dim(xin)[1] # num de dados)
  n<-dim(xin)[2] # num dimensões xin
  m<-dim(yd)[2] # num de dimensões yd

  Z<-matrix(runif((n+1)*p)-0.5, nrow = n+1, ncol = p) # dim (n+1 x p)
  W<-matrix(runif((p+1)*m)-0.5, nrow = p+1, ncol = m) # dim (p+1 x m)

  xin<-cbind(xin,1) # dim (N, n+1)

  nepocas<-0
  eepoca<-tol+1
  #inicializa vetor erro evec,
  evec<-matrix(nrow=1,ncol=maxepocas)
  while ((nepocas < maxepocas) && (eepoca>tol))#eepocas erro da epoca e tol tolerancia
  {
    ei2<-0
    #sequencia aleatoria para treinamento
    xseq<-sample(N)
    for (i in 1:N)
    {
      #padrao para sequencia aleatoria
      irand<-xseq[i]

      U<-xin[irand,]%*%Z # xin (1, n+1) %*% Z (n+1 x p) = U (1 x p)
      H<-tanh(U) # dim (1 x p)
      Haug<-cbind(H,1) # dim (1 x p+1)

      O<-Haug%*%W # Haug (1, p+1) %*% W (p+1 x m) = O (1 x m)

      if (func_ati == 0){
        yhat<-O # dim (1 x m) - Função de ativação linear
      } else {
        yhat<-tanh(O) # dim (1 x m) - Função de ativação tanh
      }

      ## Backpropagation

      ei<-yd[irand,]-yhat # ei (1 x m)
      if (func_ati == 0){
        flinhaO<-1 # Função de ativação linear
      } else {
        flinhaO<-sech2(O) # flinhaO (1 x m) - Função de ativação tanh
      }
      dO<-ei*flinhaO # dO (1 x m) - produto ei x (elemento a elemento)

      Wminus<-W[-(p+1),] # Wminus (p x m) - Removendo o term de polarização (Não se propaga)
      ehidden<-dO %*% t(Wminus) # dO (1 x m) %*% t(Wminus) (m x p) = ehidden (1 x p)
      flinhaU<-sech2(U) # flinhaU (1 x p)
      dU<-ehidden*flinhaU # dU (1 x p) - produto ehidden x (elemento a elemento)

      W<-W+eta*(t(Haug) %*% dO) # (p+1 x m) + eta * [(p+1 x 1) %*% (1 x m)] = (p+1 x m)
      xint<-t(xin)
      Z<-Z+eta*(xint[,irand] %*% dU) # (n+1 x p) + eta * [(n+1 x 1) %*% (1 x p))] = (n+1 x p)

      ei2<-ei2+(ei*t(ei))
    }
    #numero de epocas
    nepocas<-nepocas+1
    evec[nepocas]<-ei2/N
    #erro por epoca
    eepoca<-evec[nepocas]
  }
  retlist<-list(W,Z,evec[1:nepocas])
  return(retlist)
}
```