UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

ATIVIDADE AVALIATIVA
TREINAMENTO MLP - BACKPROPAGATION

ALUNO: FELIPPE VELOSO MARINHO
MATRÍCULA: 2021072260
DISCIPLINA: REDES NEURAIS ARTIFICIAIS

As MLP (MultiLayer Perceptron) ou perceptrons de múltiplas camadas, treinadas com um método iterativo baseado no gradiente descendente.
O treinamento de redes como estas (ELM e RBFs), capazes de resolverem problemas não-lineares são feitos em duas camadas:

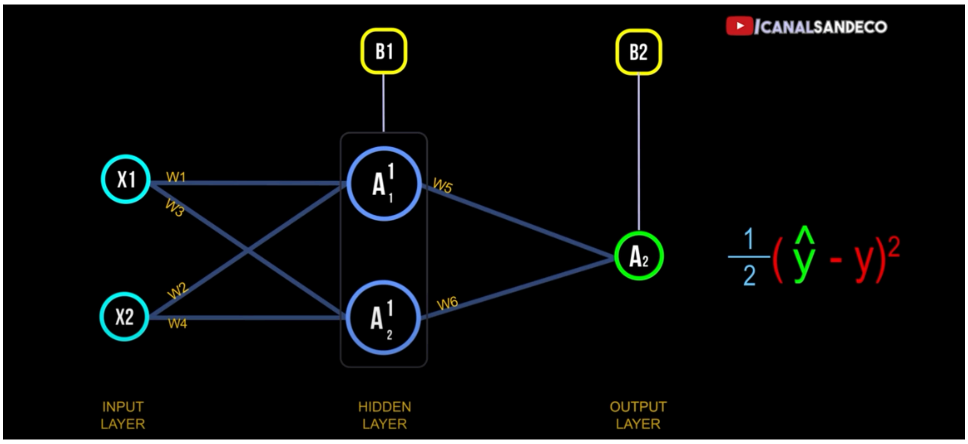
Uma camada intermediária e outra escondida. Uma vez que temos o mapeamento desse espaço de entrada para o espaço da camada escondida. Sendo assim, após esse mapeamento, temos o problema linearizado na saída. Assim podemos resolver com um neurônio como adaline, por exemplo.

Em MLPs utilizamos o método de gradiente descendente para a solução do problema e atualização dos pesos.

Então relembrando, uma rede neural é formada por camada de entrada, camadas ocultas e uma camada de saída. Por ser um Multilayer Perceptron, todos os neurônios são conectados aos neurônios da camada anterior por meios de pesos e bias. Porém, como podemos atualizar os pesos de uma rede neural?

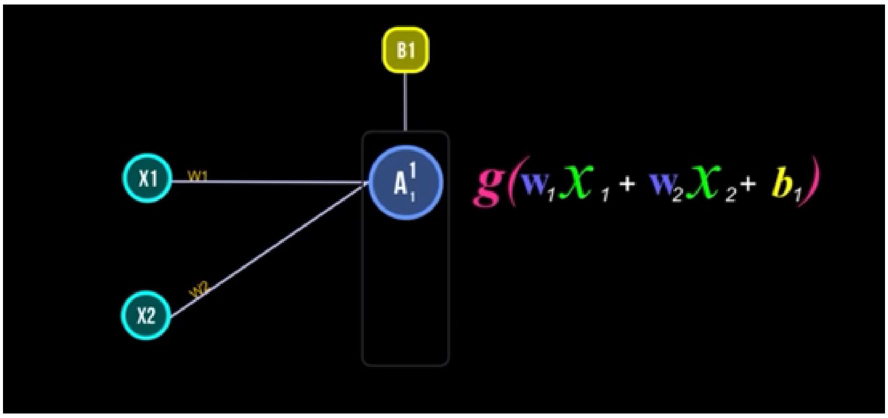
Backpropagation - Revisão da Teoria

Ele possui basicamente duas fases, a propagação e a retropropagação.

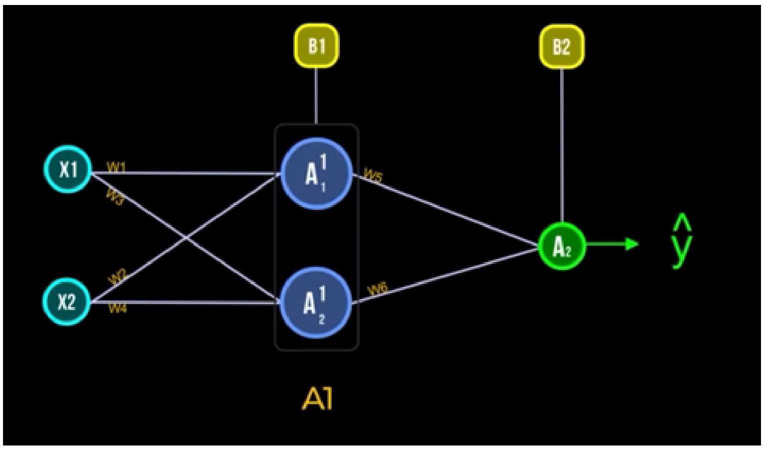


Em um problema de regressão simples, inicializamos os pesos de maneira aleatória, e no final verificamos o erro percentual. Sendo assim, propagamos esses valores até a verificação na saída. Se o erro for grande, ajustamos os pesos retropropagando o erro através da rede neural. Partindo da camada de saída, passando pelas camadas ocultas até a camada de entrada. Esse processo é repetido n vezes, até que o erro seja o menor possível.

Como o objetivo é sempre diminuir o custo em um algoritmo de otimização. Porém, devido aos pesos estarem envolvidos pelas funções de ativação (sigmóide geralmente). Para isso, em cada neurônio temos a soma ponderada envolta a função de ativação.

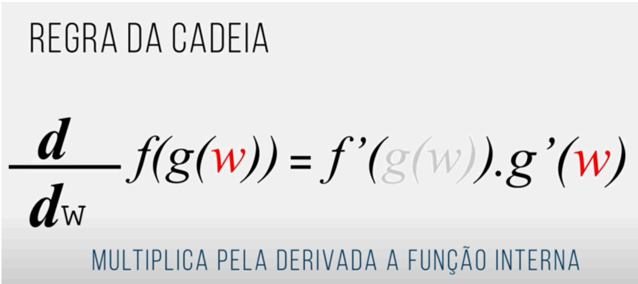


Estendendo isso para dois neurônios temos uma soma ponderada ainda maior envoltas cada uma em uma função de ativação:

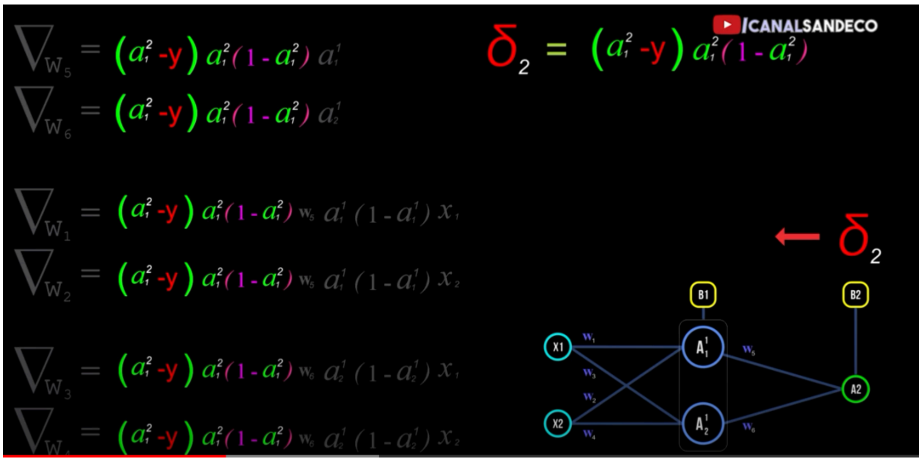


$$\hat{y} = g(w_5 a_1^1 + w_6 a_2^1 + b_2)$$

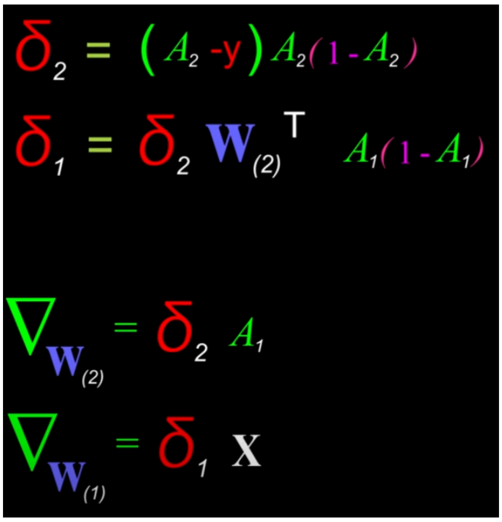
Quando derivamos o erro em relação ao peso utilizando a temível regra da cadeia:



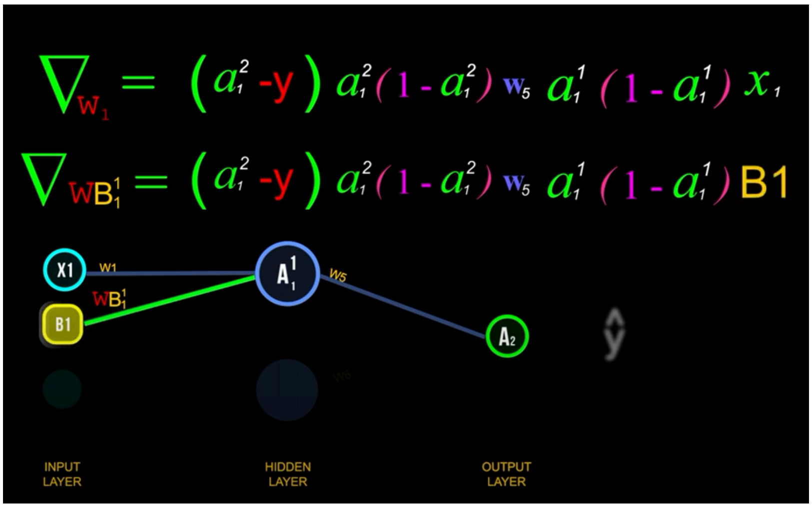
Através dela, é possível atualizar os pesos das camadas intermediárias, representadas pelos pesos w5 e w6.



Para simplificar os cálculos, a cada derivação feita encontramos o gradiente de cada peso. Nesses gradientes, é visível termos que se repetem. Esses termos iremos chamar de delta. Esse é um exemplo de uso da regra delta, calculamos os deltas e depois os gradientes de todas as camadas.



Para atualizar os pesos dos bias repetimos praticamente o mesmo caminho na retropropagação. alterando somente o finalzinho como demonstrado na imagem abaixo:



Backpropagation - Exercício Proposto

Para o exercício é necessário observar uma rede MLP realizando a regressão de um ciclo de uma senóide com backpropagation. A função de ativação escolhida deve ser linear e devem haver 3 neurônios na camada escondida.

Posteriormente devemos implementar uma função de treinamento sem utilizar pacotes, mas escrevendo as equações de atualização do neurônio de saída e dos três neurônios da camada escondida.

Deve-se calcular o erro quadrático médio (MSE) percentual do classificador.

Esse procedimento deve ser repetido 5 vezes para estimar o MSE percentual médio e o desvio padrão do classificador.

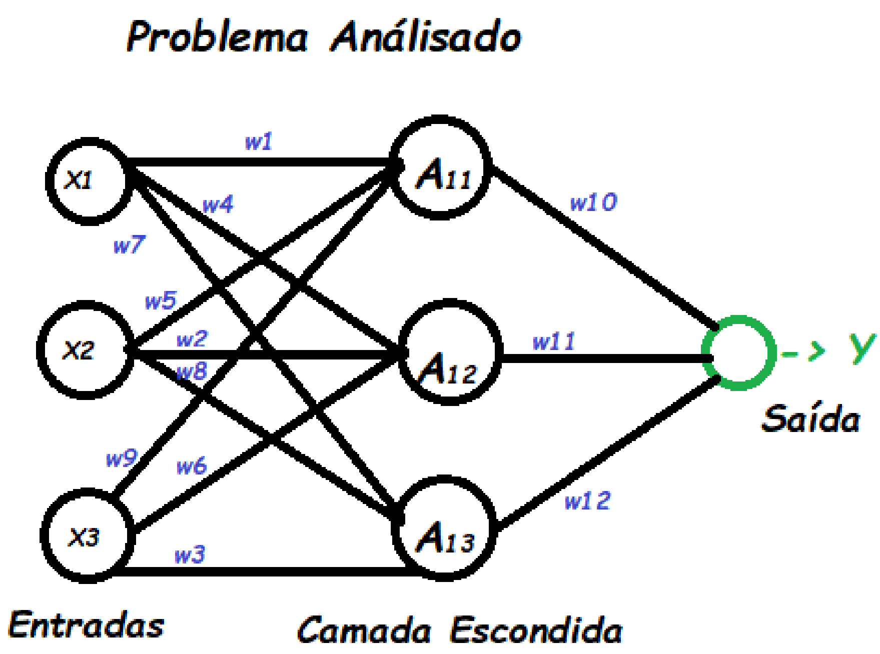
Por fim, deve-se também plotar as saídas da função aproximada e os valores esperados y de forma ilustrativa para uma única execução.

Implementação

1 - Criação das funções de ativação lineares;

1.1 - A função definida será **f(x)=x,** e sua derivação usada para o cálculo dos gradientes é simplesmente constante ou igual a 1.

2 - Função para treino da rede MLP percorridas por determinado número de epoch



2.1 - Inicializo todos os pesos e bias do problema com três neurônios inicialmente com os valores aleatórios;

2.2 - Realizo a propagação e armazeno o erro ou MSE.

2.3 - Faço a retropropagação atualizando os pesos e bias.

3 - Crio uma função para predição

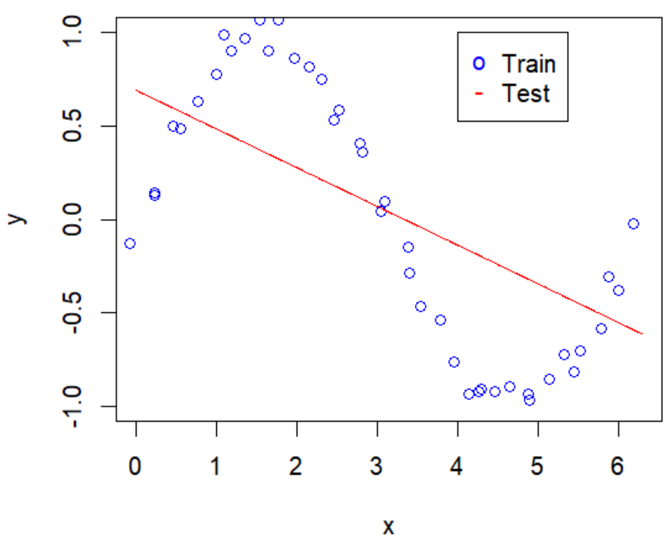
3.1 - Utilizo os pesos e bias treinados para realizar as operações.

4 - Faço um loop com as 5 vezes estimando o MSE percentual médio e o desvio padrão do classificador.

5 - Ploto as saídas da função aproximada e os valores esperados de y.

Utilizando a função de ativação como a linear **f(x)=x,** chegamos no seguinte resultado:

```
Erro médio quadrático (MSE): 0.2286254
```



Rodando mais 4 vezes podemos verificar a média de MSE:

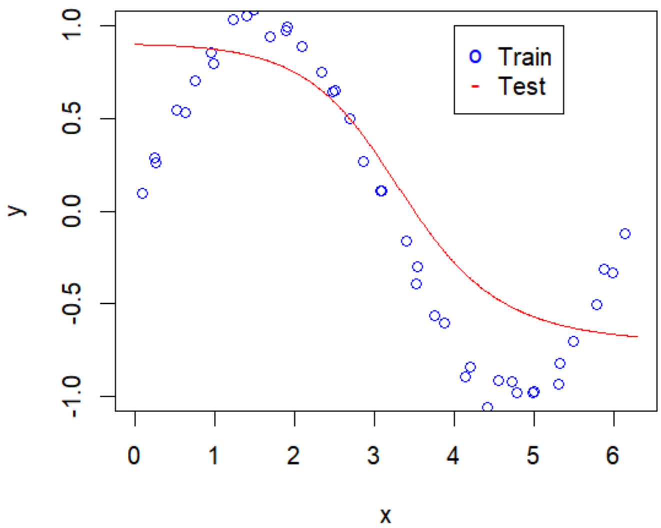
```
Erro médio quadrático (MSE): 0.2000474
Erro médio quadrático (MSE): 0.2002207
Erro médio quadrático (MSE): 0.2006071
Erro médio quadrático (MSE): 0.3017039
Erro médio quadrático (MSE): 0.2166944
```

O erro médio quadrático (MSE) médio das 5 execuções da rede neural é aproximadamente **0.2246** com um desvio padrão de **0.0360.**

Verificando o gráfico, como extra, foi alterada a função de ativação para melhorar a aproximação da rede neural ao problema da função senoidal. Testando com a função tangente hiperbólica (tanh) que mapeia os valores de entrada para o intervalo [-1, 1] temos um comportamento mais adequado ao seno.

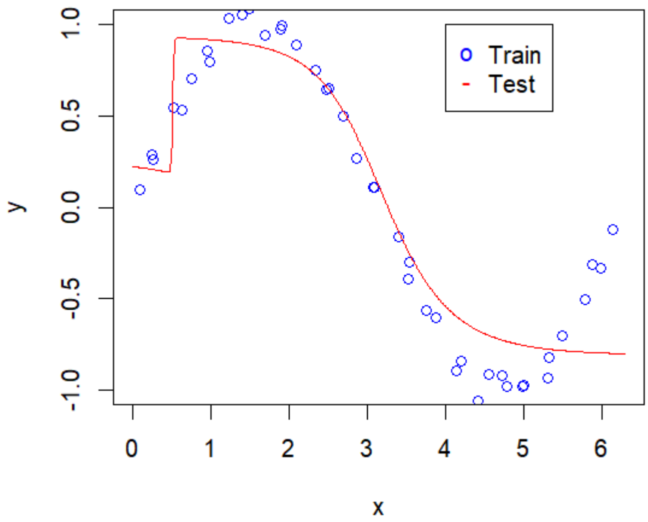
Com ela temos uma aproximação bem maior dos dados de treino e os de teste com o erro médio quadrático sendo bem menor que quando utilizada a função de ativação linear.

```
Erro médio quadrático (MSE): 0.124347
```



Testando com ainda mais ephocs (10000) chegamos a um resultado bem mais próximo da curva de treino com um erro médio ainda mais baixo:

```
Erro médio quadrático (MSE): 0.05210788
```



Código Completo

```
# Função de ativação linear
linear <- function(x) {
  return(x)
}

# Derivada da função de ativação linear
linear_derivative <- function(x) {
  return(1)
}

# # Função de ativação tanh
# tanh_activation <- function(x) {
#   return(tanh(x))
# }
#
# # Derivada da função de ativação tanh
# tanh_derivative <- function(x) {
#   return(1 - tanh(x)^2)
# }

#Inicializando dados
x_train <- seq(from = 0, to = 2*pi, by = 0.15)
x_train <- x_train + (runif(length(x_train)) - 0.5)/5

i <- sample(length(x_train))

x_train <- x_train[i]
y_train <- sin(x_train)
y_train <- y_train + (runif(length(y_train)) - 0.5)/5

plot(x_train, y_train, col = 'blue', xlim = c(0, 2*pi),
```

```
     ylim = c(-1, 1), xlab = 'x', ylab = 'y')

x_test <- seq(from = 0, to = 2*pi, by = 0.01)
y_test <- sin(x_test)

par(new = T)

plot(x_test, y_test, col = 'red', type = 'l', xlim = c(0, 2*pi),
     ylim = c(-1, 1), xlab = 'x', ylab = 'y')

legend(x = 4, y = 1, legend = c('train', 'test'),
       col = c('blue', 'red'), pch = c('o', '_'))


# Inicialização dos pesos
input_neurons <- 1
hidden_neurons <- 3
output_neurons <- 1

# Pesos
w1 <- matrix(runif(input_neurons * hidden_neurons), nrow = input_neurons)
w2 <- matrix(runif(hidden_neurons * output_neurons), nrow = hidden_neurons)
b1 <- matrix(runif(hidden_neurons), nrow = 1)
b2 <- matrix(runif(output_neurons), nrow = 1)


# Taxa de aprendizado
learning_rate <- 0.075

# Número de épocas (qtd de iterações)
epochs <- 10

# Treinamento da rede
for(epoch in 1:epochs) {
  for(i in 1:length(x_train))
  {
    # Feedforward (propagação direta)
    input_data <- matrix(x_train[i], nrow = 1)
    hidden_input <- input_data %*% w1 + b1
    hidden_output <- linear(hidden_input) # ativação da camada escondida

    output_input <- hidden_output %*% w2 + b2
    output <- linear(output_input) # saída final da rede neural

    # Cálculo do erro
    error <- y_train[i] - output

    # Backpropagation
```

```
    output_delta <- error * linear_derivative(output) # Calculo do delta da camada de saída,
que nos diz o quanto temos que ajustar
    # os pesos da camada de saída para reduzir o erro.

    hidden_error <- output_delta %*% t(w2) # Cálculo do erro retropropagado da camada
escondida.Temos que transpor os pesos
    # que conectam a camada escondida a camada de saída por conta de ajuste a dimensão.

    hidden_delta <- hidden_error * linear_derivative(hidden_output) # Cálculo do delta da
camada escondida, que nos diz o quanto precisamos ajustar
    # os pesos da camada escondida para reduzir o erro.

    # Atualização dos pesos
    w1 <- w1 + t(input_data) %*% hidden_delta * learning_rate
    w2 <- w2 + t(hidden_output) %*% output_delta * learning_rate
    b1 <- b1 + hidden_delta * learning_rate
    b2 <- b2 + output_delta * learning_rate
  }
}

# Previsões da rede treinada
predictions <- numeric(length(x_test))
for(i in 1:length(x_test))
{
  input_data <- matrix(x_test[i], nrow = 1)
  hidden_input <- input_data %*% w1 + b1
  hidden_output <- linear(hidden_input)
  output_input <- hidden_output %*% w2 + b2
  predictions[i] <- linear(output_input)
}

# Plot dos resultados
plot(x_train, y_train, col = 'blue', xlim = c(0, 2*pi),
     ylim = c(-1, 1), xlab = 'x', ylab = 'y')

lines(x_test, predictions, col = 'red') # desenha uma linha no gráfico, conectando os pontos
representados por x_test e predictions.
# isso representa as previsões da rede em relação aos dados de teste.

legend(x = 4, y = 1, legend = c('Train', 'Test'),
       col = c('blue', 'red'), pch = c('o', '-'))

# Cálculo do erro médio quadrático
mse <- mean((predictions - y_test)^2)
cat("Erro médio quadrático (MSE):", mse, "\n")
```