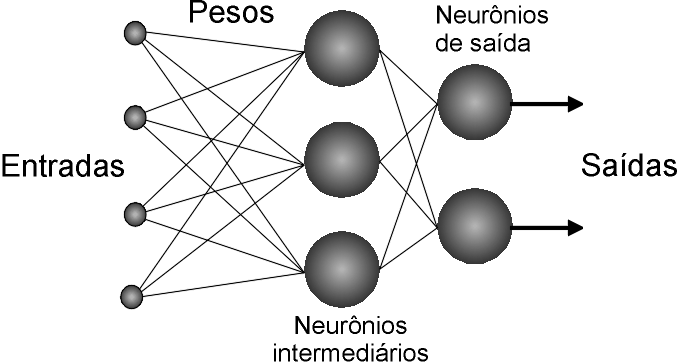
# Cálculo de Pesos e Erros

1 – O Algoritmo inicializa os pesos com valores aleatórios. Veja a figura para ver o papel dos Pesos dentro de uma Rede Neural.

Quanto maior a quantidade de entradas, maior o tempo de processamento de entradas e saídas, por conta disto, segundo o professor do curso da *Udemy*, estes sistemas são executados em servidores especializados para isto. Por exemplo, dada 50 entradas, será avaliada todas suas saídas e valores da camada oculta.



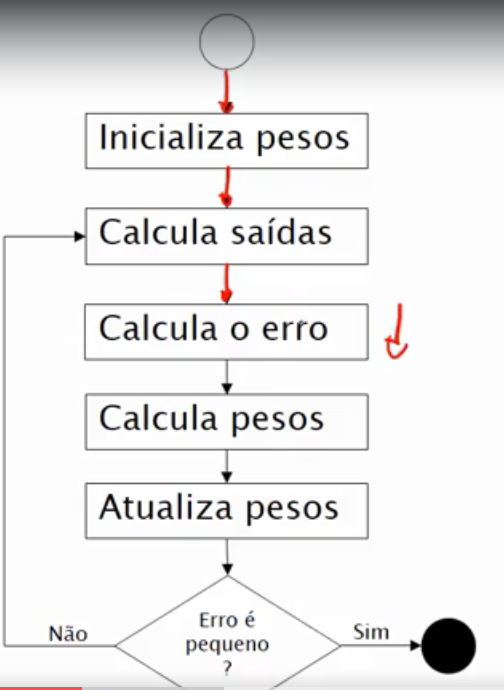
2 – Baseando se nos dados (aprendizagem supervisionada, pois já sabemos qual é a saída correta), realiza os cálculos com os pesos e calcula o erro.

Ex: Resposta Correta = 1, Resultado = 0,5. Eu queria chegar no 1 e eu adquiri 0,5 então o cáculo do erro é ( 1 – 0,5 ) = 0,5. Se o que eu adquiri for negativo, ele é colocado como positivo na hora da subtração (Abs).

3 – Calcula as mudanças nos pesos e os atualiza (Processo de *Backpropagation*).

4 – O algoritmo termina quando a taxa de erro for pequena, ou seja, o algoritmo não termina quando necessariamente a taxa de erro for ZERO.

Resumindo como é feito o Cálculo de erros e Pesos, aqui está o diagrama de blocos:

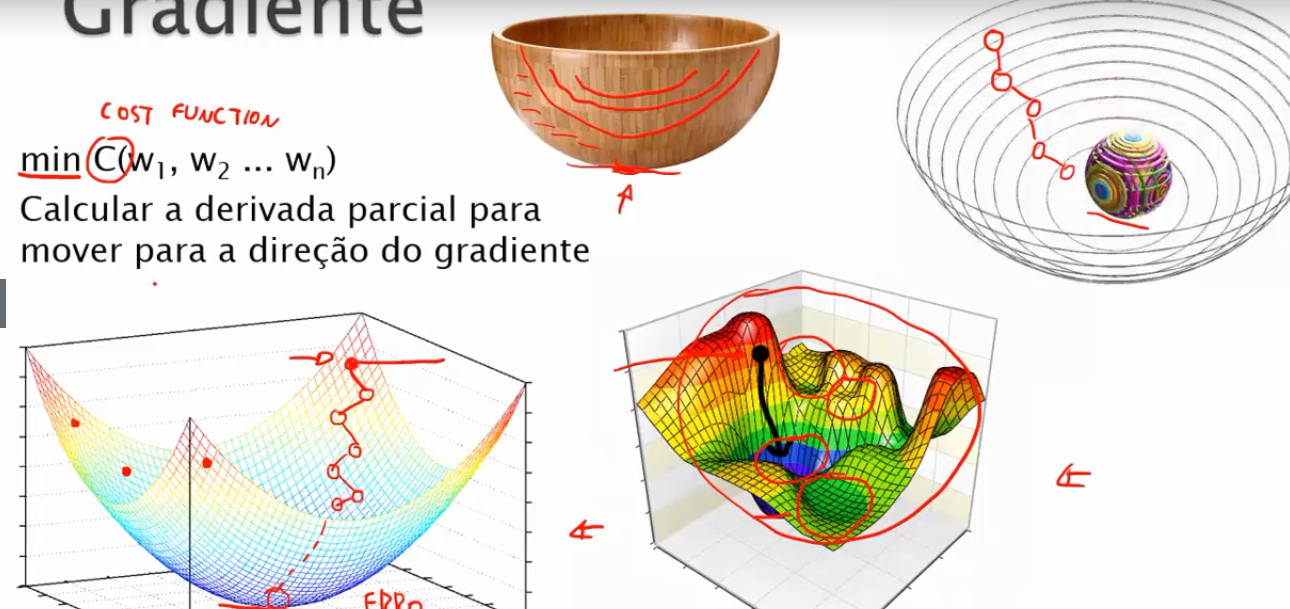


*Cost Function* – É a função de minimizar o Erro, ocorre na hora de calcular o erro.

* Para a atualização de pesos, é utilizada:
* Gradiente (*gradient descent*)
* Derivada
* Cálculo do Delta
* *BackPropagation*
* *Learning Rate* (Taxa de Aprendizagem)
* *Momentum* (Momento)

**Gradiente (*gradiente descent*)**

A finalidade é encontrar a combinação de pesos que o erro é o menor possível. Gradiente é calculado para saber **quanto ajustar os pesos.**



Força Bruta

- Fazer todas as combinações possíveis. Porém nada viável atualmente, por conta do grande tempo que vai ser levado para encontrar uma combinação de peso com o menor erro possível. Apesar dele trazer resultados mais corretos.

*Simmulated Anealing*

Algoritmos genéticos

- Algoritmos de otimização para encontrar valores. Muito conhecido na área de inteligência artificial.

**Cálculo da Derivada**

Derivada = Y \* ( 1 – Y )

**Cálculo do Delta da Saída – Aula 33**

Lembrando que a etapa: Gradiente, Derivada e Cálculo do Delta são etapas praticamente juntas.

Partindo o valor do delta, podemos fazer o cálculo do gradiente.

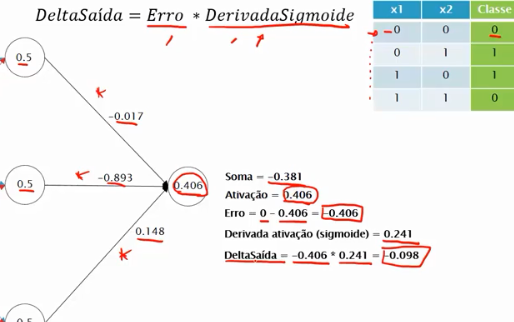
O cálculo do parâmetro Delta ocorre:

Função de Ativação -> Calcula a derivada da função -> Calculo do Parâmetro do Delta -> Cálculo do Gradiente

Para fazer seu cálculo, devemos fazer o cálculo do Delta da camada de saída e depois Delta da camada Oculta.

O Delta de saída é:

DeltaSaída = Erro \* Derivada de Sigmoide

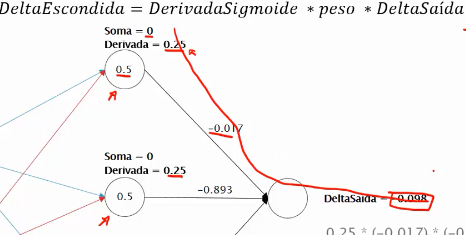


**Cálculo do Delta da Camada Oculta – Aula 35**

O delta da camada oculta tem a mesma finalidade do Delta de saída, que é obter o **gradiente**.

A sua fórmula é:

DeltaEscondida = Derivada de Sigmoide \* peso \* DeltaSaída



***Backpropagation* – Aula 37**

Retro propagação, propagar a atualização dos pesos da camada de saída para todas as outras camadas que seguem a rede neural. A fórmula para *backpropagation* é:

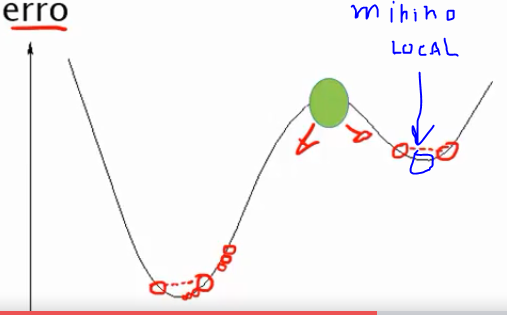
Peso n + 1 = (Peso \* momento) + (entrada \* delta \* taxa de aprendizagem)

O parâmetro da *learning rate* (Taxa de aprendizagem):

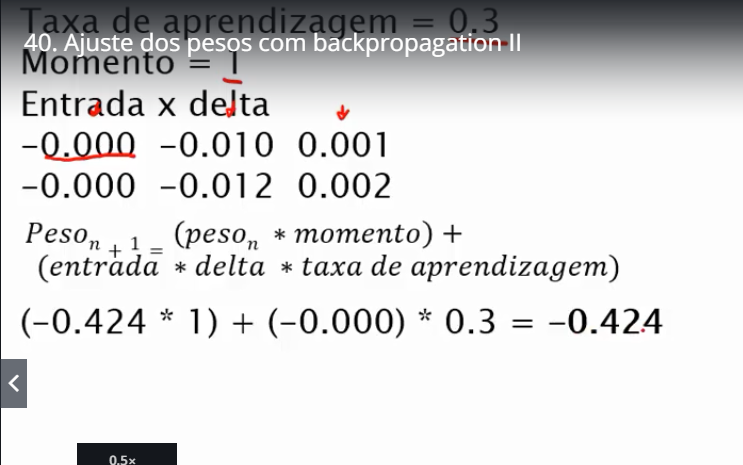
* Define quão rápido o algoritmo vai aprender
* Se o valor for Alto: A convergência é rápida mas pode perder o mínimo global
* Se o valor for Baixo: Será mais lento mas a chance de chegar no mínimo global é maior

O parâmetro *momentum* (Momento):

* Sua ideia é de escapar de mínimos locais (nem sempre funciona). E o que seria o mínimo local?

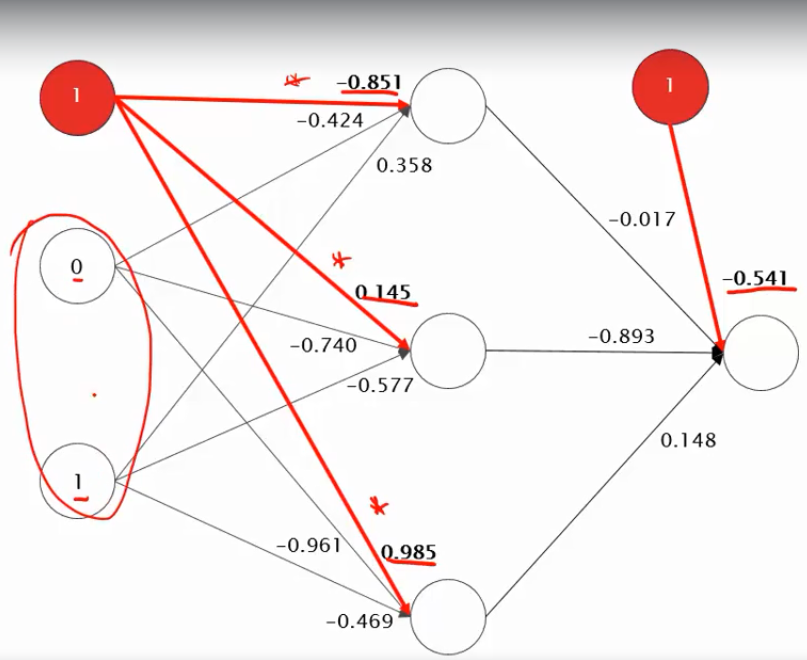


* Define o quão confiável é a última alteração
* Se o momentum for Alto: Aumenta a velocidade da convergência
* Se o momentum for Baixo: Pode evitar mínimos locais



# Bias / Viés

Temos nossa rede neural multicamada, composta pela camada de entrada, oculta e saída, a idéia do *Bias* adiciona 1 atributo/valor em cada uma das camadas (Com pesos respectivos também) e a *Bias* na camada de saída.



# Erro Mean Square Error (MSE) e Root Mean Square Error (RMSE)

# Deep Learning

* Em 2006 foi criado algoritmos para treinamento de redes neurais.
* É chamado de Deep Learning quando um algoritmo possui 2 ou mais camadas ocultas, se tornando uma rede Neural profunda segundo vários pesquisadores.
* Para seu desenvolvimento são utilizados outras técnicas (diferentes das citadas anteriormente nas anotações).
* Em redes neurais profundas, há o problema do ***vanishing gradient problem***, onde o gradiente fica muito pequeno e as mudanças nos pesos ficam pequenas.
* É utilizado outras funções de ativação
* Através do Deep Learning pode seguir: Redes neurais convolucionais, recorrentes, Keras, Theano, TensorFlow e conceito de desenvolvimento em GPU.