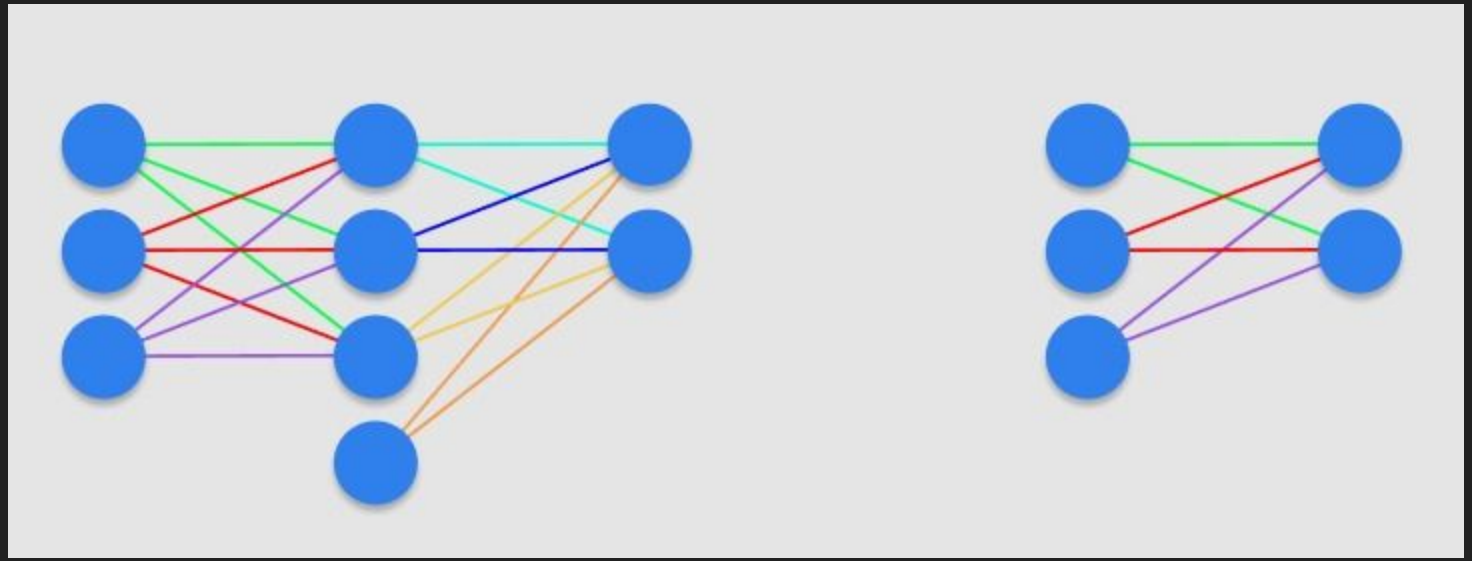
Inteligencia Artificial

Arthur Prince de Almeida - 10782990
Gabriel Urdiale - 10414612
Jose Luiz Assumpcao de Oliveira - 10687576
Mauricio Mori Dantas Santana - 7991170

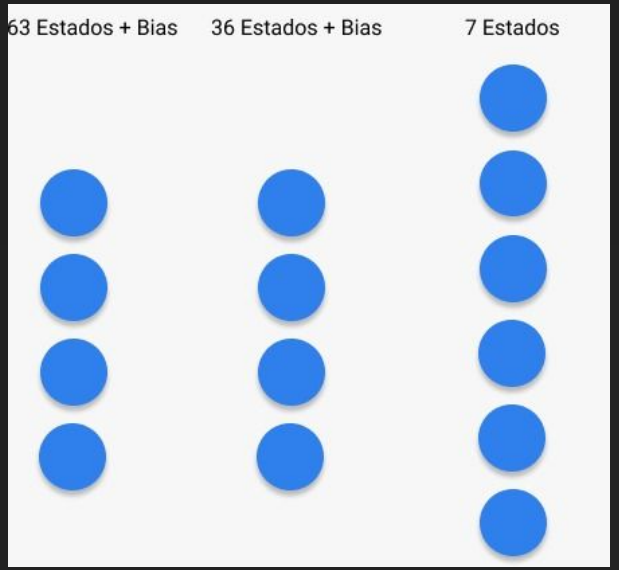
MLP

```
//construtor
public MLP(Dados dados, int nCamadas) {
    int nEntrada = dados.getEntradas(0).length;
    int nSaida = dados.getResultados(0).length;
    this.matrizDeConfusao = new int[nSaida][nSaida];
    this.nCamada=nCamadas;
    this.camadas = new Camada[nCamadas];
    this.iteracoes = dados.getIteracoes();
    this.epocaDoTxt = dados.getIteracoes();
    this.acertos = new LinkedList<Integer>();
    this.taxaDeAprendizado = dados.getTaxaDeAprendizado();

    this.camadas[nCamadas-1]= new Camada(nSaida);
    //a media e usado para definir quantos estados tem em cada camada escondida
    int media = (int)(nEntrada+nSaida)/(nCamadas-1)+1;
    //coloca estados nas camadas escondidadas de acordo com a media aritmetica

    int i= nCamadas-3;
    //faz a camada anterior a de saida ter uma funcao linear e as outras uma funcao
    //nao linear
    if(i>=0) {
        this.camadas[i+1] = new EntradaOuEscondida(media,this.camadas[i+2],true);
        while(i>0) {
            this.camadas[i] = new EntradaOuEscondida(media,this.camadas[i+1],false);
            i++;
        }

        this.camadas[0]=new EntradaOuEscondida(nEntrada,this.camadas[1],false);
    }
    else
        this.camadas[0]=new EntradaOuEscondida(nEntrada,this.camadas[1],true);
}
```

MLP

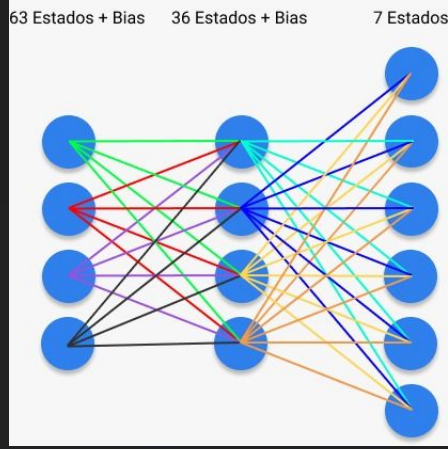


Camadas - Codigo




```
public class Camada {
    private double[] states;
    private int nstates;

    //construtor que carrega as variaveis
    public Camada(int numeroDeEstados) {
        this.nstates=numeroDeEstados;
        this.states = new double[numeroDeEstados];
    }
```

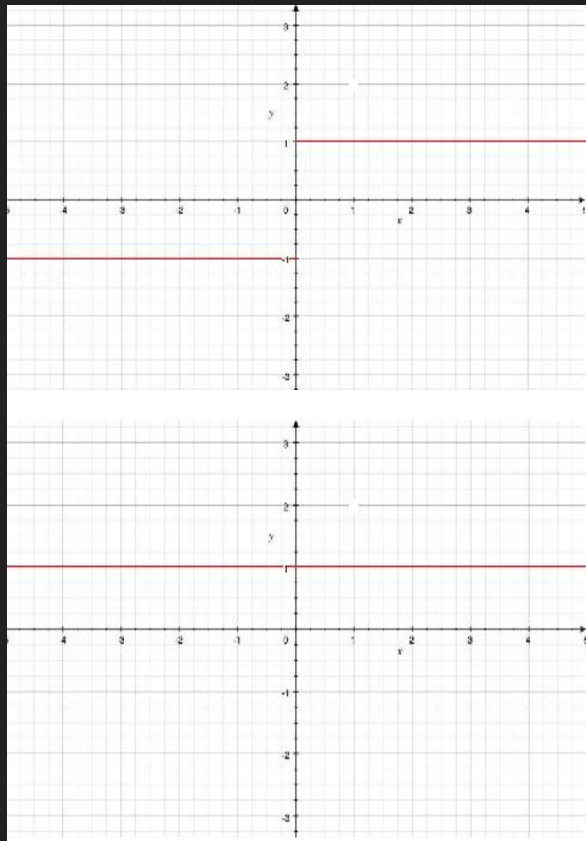
Camadas - Codigo




```java
public class EntradaOuEscondida extends Camada {

    private double[][] weight;
    public Funcao funcao;

    /*
     * essa camada tem peso e tipo de funcoes para calcular a ativacao dos estados e os erros
     */
    public EntradaOuEscondida(int numeroDeEstados, Camada vizinho, boolean penultimaCamada) {
        super(numeroDeEstados);

        this.weight= new double[numeroDeEstados+1][vizinho.getNumeroDeStates()];
        if(penultimaCamada)
            this.funcao= new Linear();
        else
            this.funcao=new NLinear();
    }
```

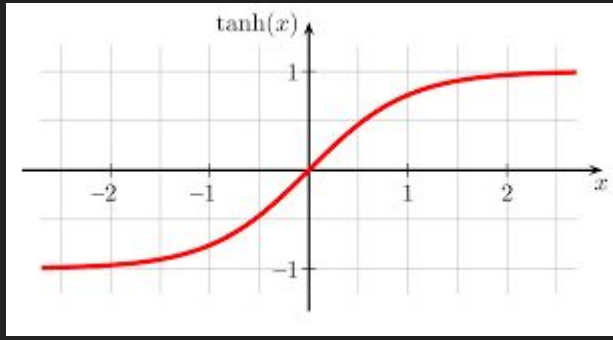
FUNÇÕES - LINEAR

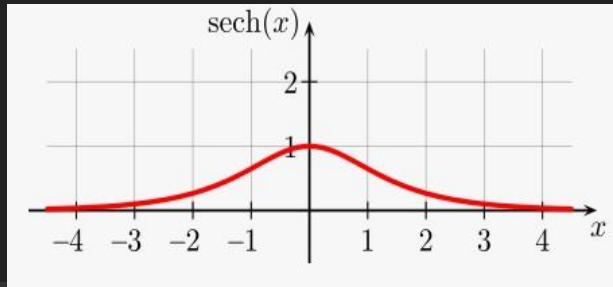
```java
public class Linear implements Funcao {

    public double funcao(double x) {
        if(x>0) {
            return 1;
        }
        return -1;
    }

    public double derivadaFuncao(double x) {
        return 1;
    }

    public double funcaoDeErro(double[] corretor, Camada atual, int k) {
        double[] estados= atual.getStates();
        return (corretor[k]-estados[k])/2;
    }

}
```



FUNÇÃO - NÃO LINEAR

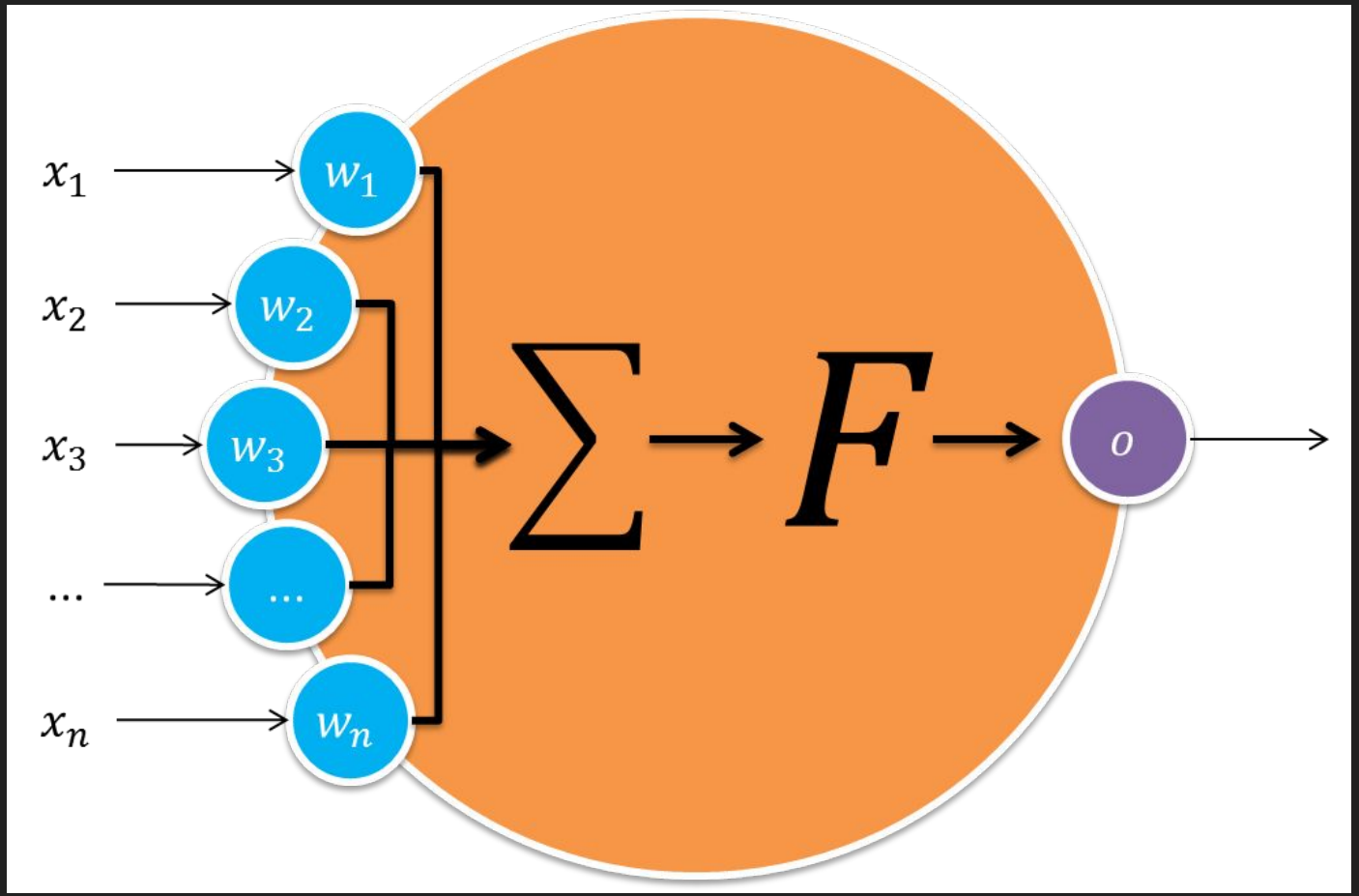
```java
public class NLinear implements Funcao {

    public double funcao(double x) {
        // tangente hiperbolica
        return Math.tanh(x);
    }

    public double derivadaFuncao(double x) {
        // sech^2(x)
        return Math.pow(1/Math.cosh(x), 2);
    }

    public double funcaoDeErro(double[] corretor, Camada atual, int k) {
        EntradaOuEscondida camada = (EntradaOuEscondida) atual;
        double[][] pesos = camada.getWeight();
        double rtn=0;
        for (int i = 0; i < corretor.length; i++) {
            rtn += corretor[i]*pesos[k][i];
        }
        return rtn;
    }

}
```





Função de ativação

```
private void ativandoEstados() {
    EntradaOuEscondida c;
    Camada prox;
    double[] novosEstados;

    for(int i = 0 ; i<this.nCamada-1;i++) {
        c = (EntradaOuEscondida)this.camadas[i];
        prox = this.camadas[i+1];
        novosEstados = new double[prox.getNumeroDeStates()];

        // Estado atual = Combinacao linear dos estados com peso
        for(int j = 0; j<prox.getNumeroDeStates();j++) {
            novosEstados[j] = c.funcao.funcao(c.combinacaoLinear(j));
        }

        prox.setStates(novosEstados);
    }
}
```


x_1
x_2
x_3
...
x_n
w_1
w_2
w_3
...
w_n
$\sum$
F
o

Treinamento

```
//treina a mlp
public void treinamento(double[][][] pesos, Dados dados) {
    if(!carregaPesos(pesos))
        return;

    //variaveis
    int[][] entradas = dados.getEntradas();
    int[][] resultados = dados.getResultados();
    int contador = 0;
    int acertos=0;
```

Treinamento - Criterio de parada

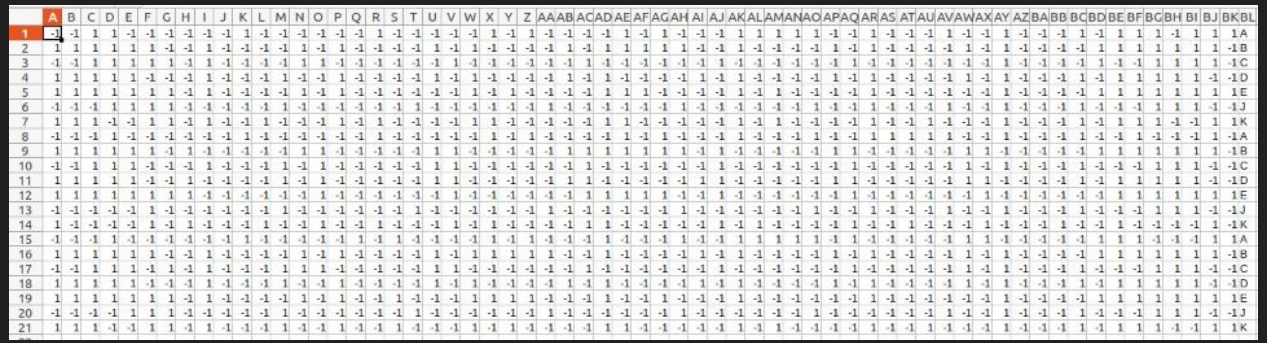
```
//so para se acertar todos os resultados ou iterar o numero limite de vezes
while(contador<this.iteracoes && acertos!=resultados.length) {
    //reseta a matriz de confusao
    for(int i = 0; i<this.matrizDeConfusao.length;i++) {
        for (int j = 0; j < this.matrizDeConfusao.length; j++) {
            this.matrizDeConfusao[i][j] = 0;
        }
    }
    contador++;
    acertos=0;
    for (int i = 0; i < resultados.length; i++) {
        carregaEntrada(entradas[i]);
        ativandoEstados();
        dados.escreveSaidas(this.camadas[this.nCamada-1].getStates(), contador, false);
        double[][][] erros = calculaErros(resultados[i], taxaDeAprendizado);
        dados.escreveErro(erros, contador, false);

        for (int j = 0; j < erros.length; j++) {
            atualizaPesos(erros, j);
        }
        //confirma se a mlp acertou o resultado do teste
        if(acertouResultado(resultados,this.camadas[this.nCamada-1].getStates(),i))
            acertos++;
    }
    this.acertos.add(acertos);
}
```

Exemplos de entradas e resultados

AND

	Padrão	Padrão	Padrão
1	-1	-1	0
2	-1	1	0
3	1	-1	0
4	1	1	1

Caracteres Limpos



Cálculo de Erro

```
private double[][][] calculaErros(int[] resultados,double taxaDeAprendizado){
    int camada = this.nCamada-1;
    double[][][] erros = new double[camada][][];

    double[] correcao = new double[resultados.length];
    for(int i =0;i<resultados.length;i++)
        correcao[i]=(double)resultados[i];
```

Cálculo de Erro

```
while(camada>0) {
    //salva a camada atual e a anterior
    Camada atual = this.camadas[camada];
    EntradaOuEscondida anterior = (EntradaOuEscondida) this.camadas[camada-1];
    //arruma espaco na matriz de erro para os erros dos pesos com o bias
    erros[camada-1]= new double[anterior.getNumeroDeStates()+1][atual.getNumeroDeStates()];
    //usado para salvar o lambidaK
    double[] atualizador = new double[atual.getNumeroDeStates()];
    double lambidaK;
    int k=0;

    double[] estadosAnterior = anterior.getStates();
    //calcula o erro para cada estado de "atual"

    while(k<atual.getNumeroDeStates()) {

        //(Tk-Yk)*f'(Yk_IN) ou somaj(Wkj*correcaoj)*f'(Zk_IN) caso nao esteja a penultima camada
        lambidaK = anterior.funcao.funcaoDeErro(correcao, atual, k)*anterior.funcao.derivadaFuncao(anterior.combinacaoLinear(k)
        //salva o lambidaK para ajudar na correcao da proxima camada
        atualizador[k]= lambidaK;

        //carrega os erros dos pesos

        for (int j = 0; j < anterior.getNumeroDeStates(); j++) {
            //dentaW= alfa*lambidaK*(estado anterior j)

            erros[camada-1][j+1][k]=taxaDeAprendizado*lambidaK*estadosAnterior[j];

        }
        //bias lambidaK*alfa
        erros[camada-1][0][k]=taxaDeAprendizado*lambidaK;
        k++;
    }
    //atualiza o corretor e volta uma camada
    correcao = atualizador;
    camada--;

}
```