



UNINASSAU

REDES NEURAIS ARTIFICIAIS

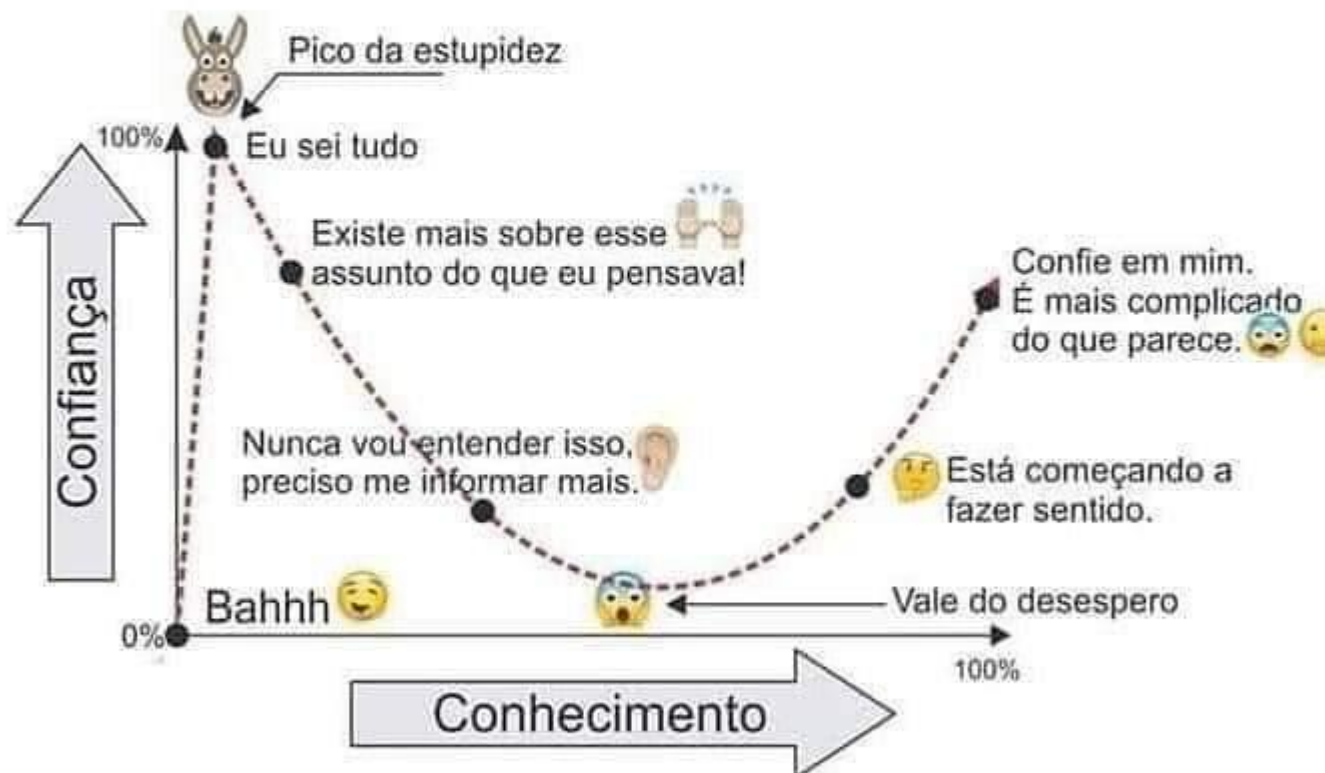
Professor: André Guerra



“Há três caminhos para o fracasso: Não ensinar o que se sabe, não praticar o que se ensina, não perguntar o que se ignora.”

São Beda – 673 - 735

Efeito Dunning-Kruger





Perceptron e ADALINE: Fundamentos e Treinamento

Objetivos da Aula:

- 1 – Entender o conceito de Perceptron
- 2 – Explorar o treinamento do Perceptron
- 3 – Introdução do ADALINE (*Adaptative Linear Neuron*)
- 4 – Compreender a regra Delta



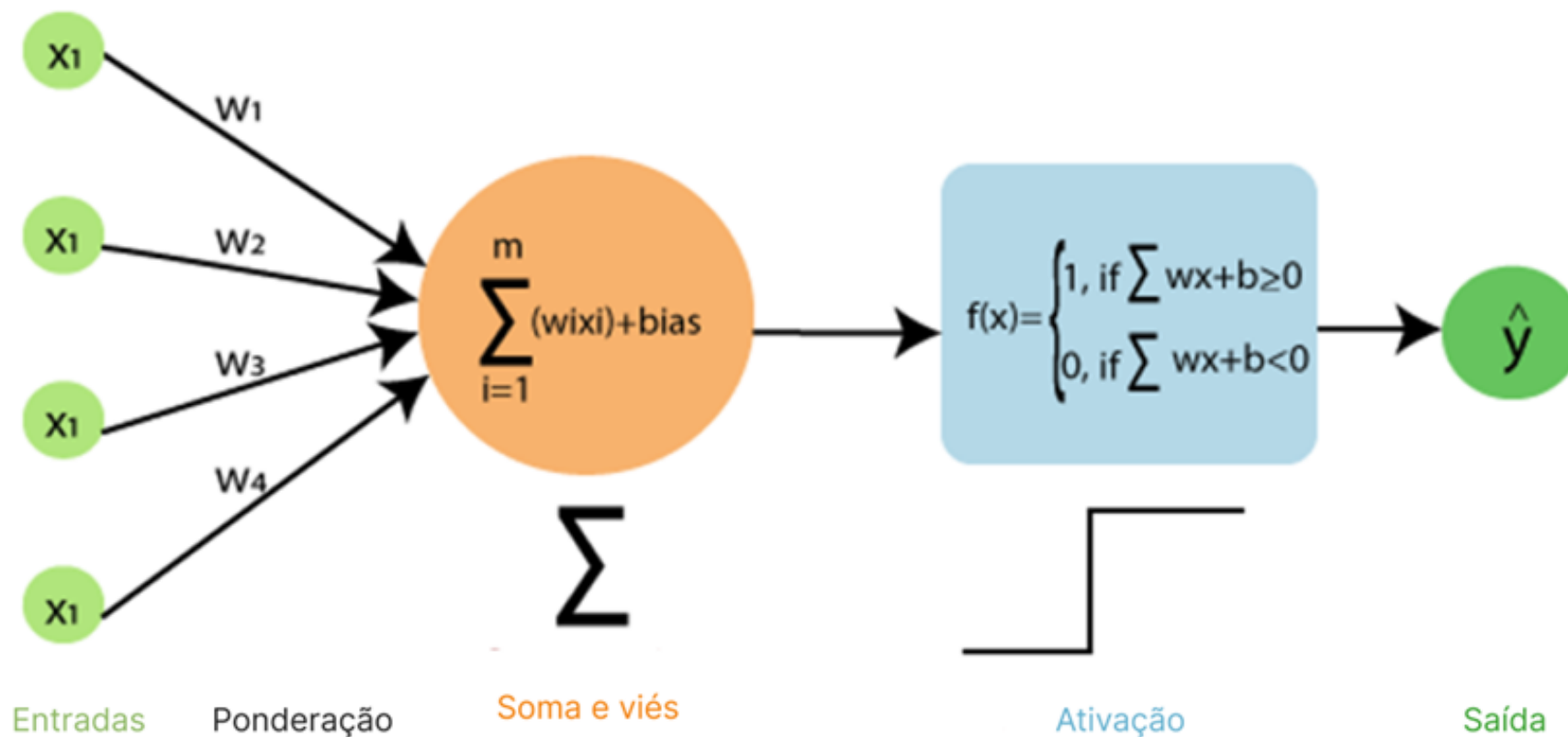
O Perceptron

- Conceito criado em 1957 por **Frank Rosenblat**
- Unidade básica de um classificador **LINEAR**

Componentes

- Entrada (**X**)
- Pesos (**W**)
- Soma ponderada(**$W_i X_i$**)
- Função de ativação – Função degrau, ativada ou não a partir de um limiar

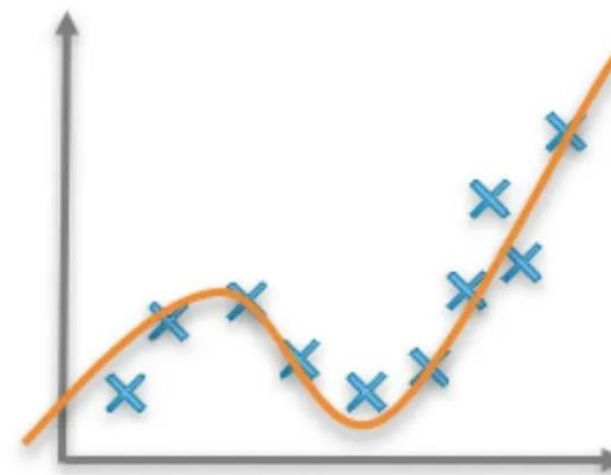
Funcionamento do perceptron



Função linear



Linear function

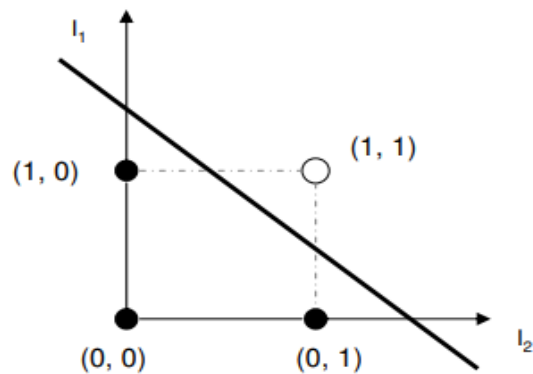


Non-linear function

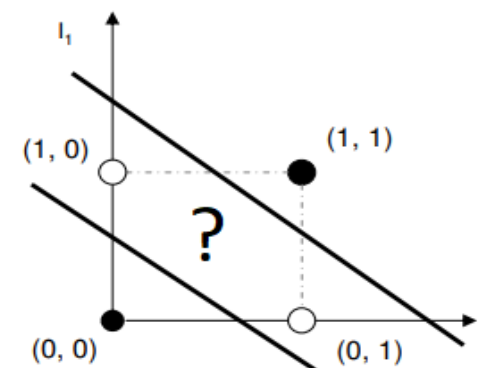
Best fit linear and non-linear models

Perceptron de camada unica

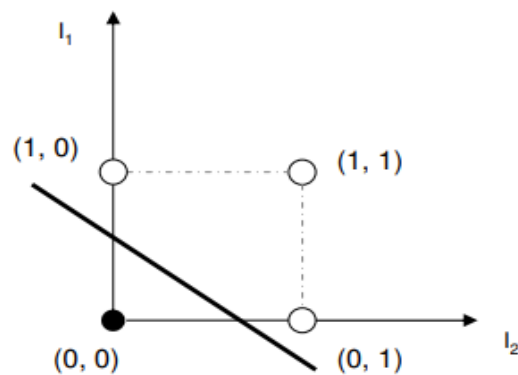
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



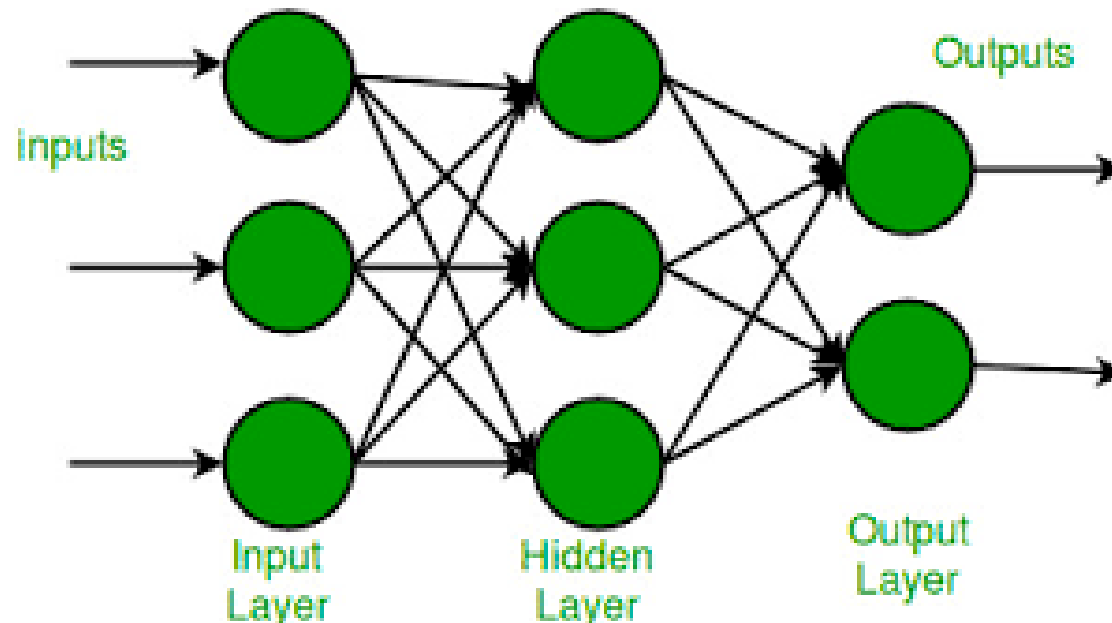
OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



Limitação em resolver problemas não lineares

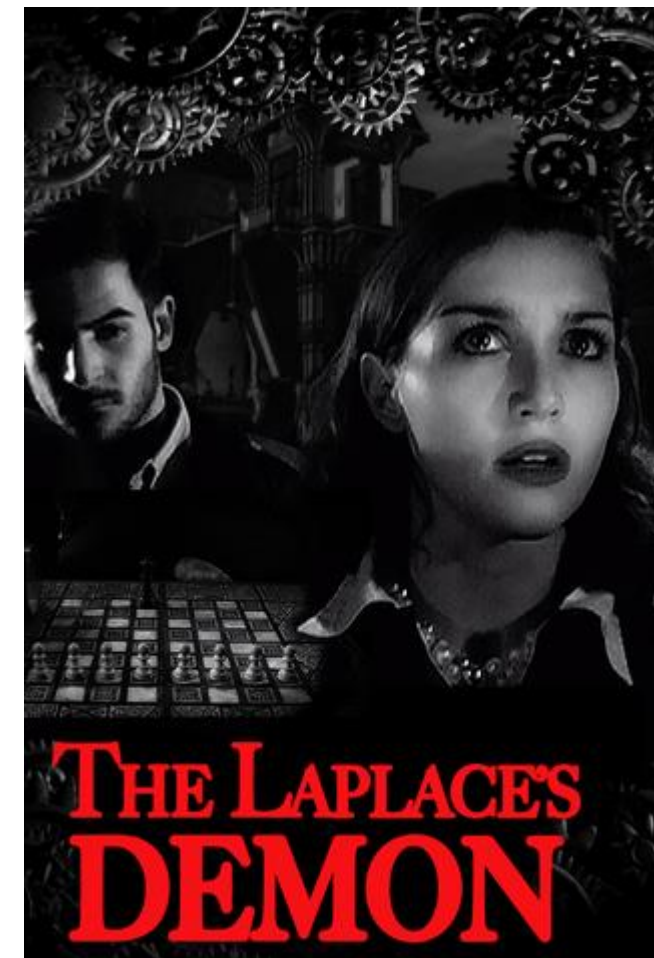
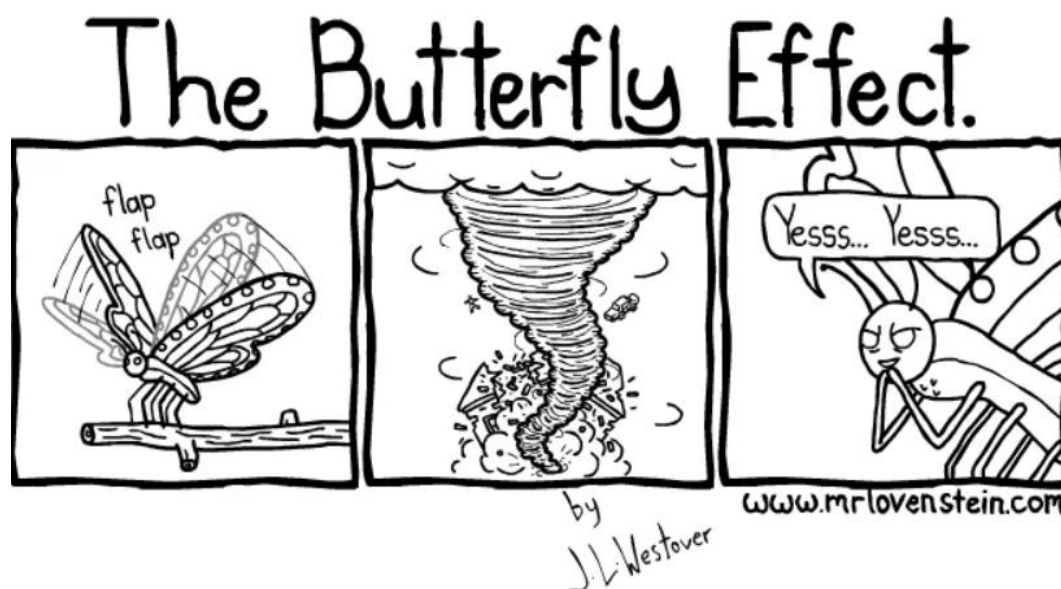
Perceptron de Múltiplas camadas (MLP)

- Introduziu o conceito de camadas ocultas
- Possibilita resolução de problemas não lineares



O demônio de Laplace

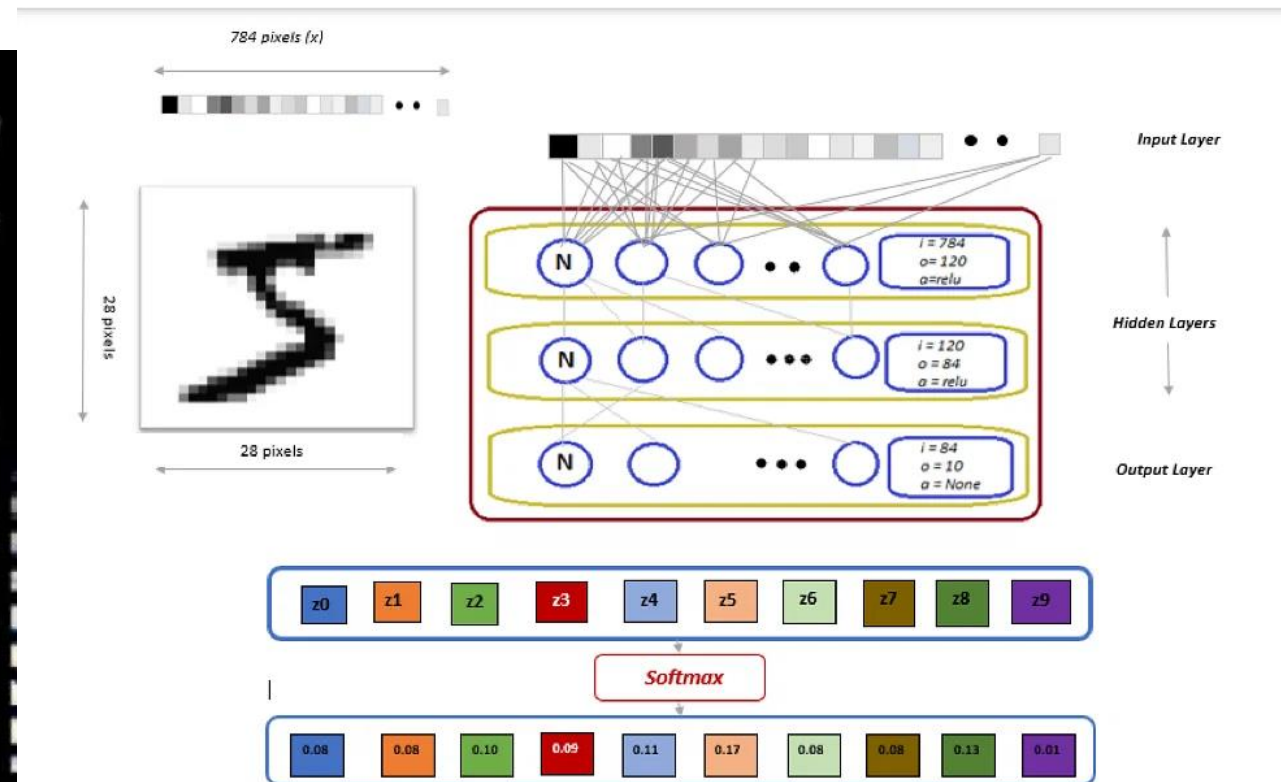
“De posse de todas as variáveis que determinam o estado do universo em um instante t ele pode prever o seu estado no instante t_1 ”



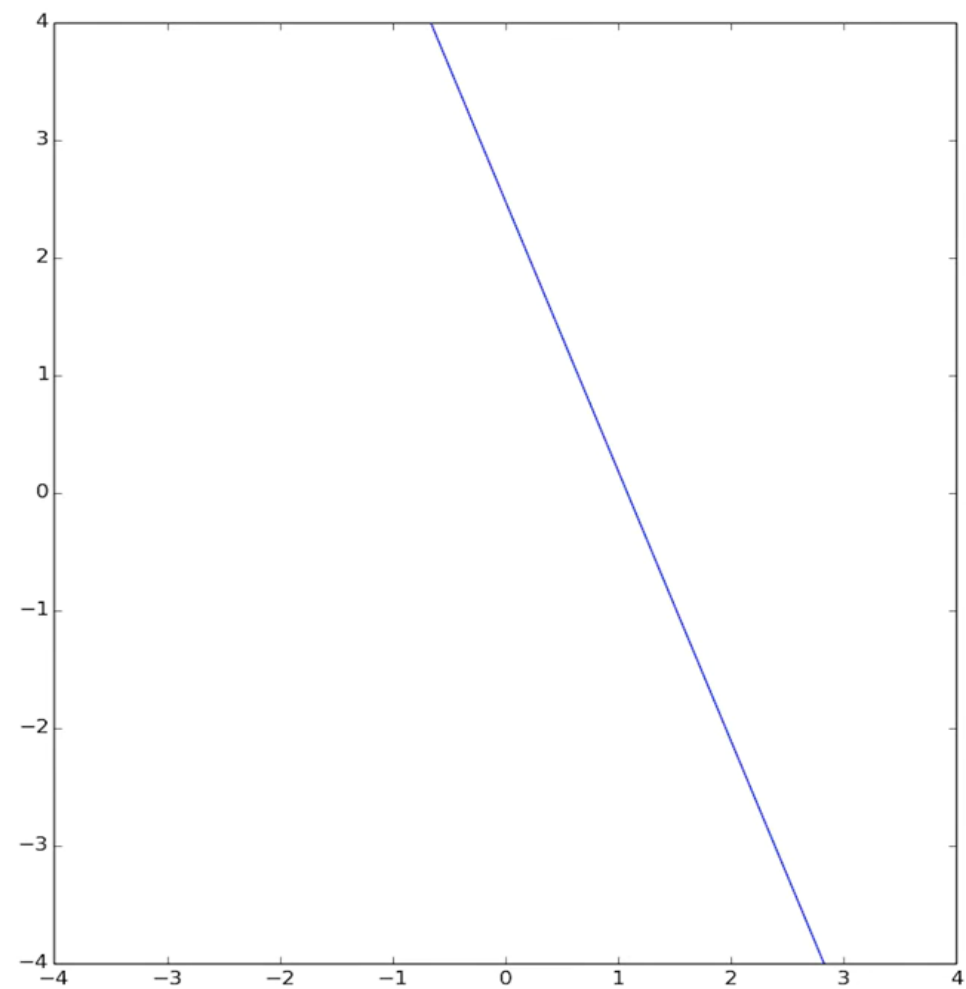
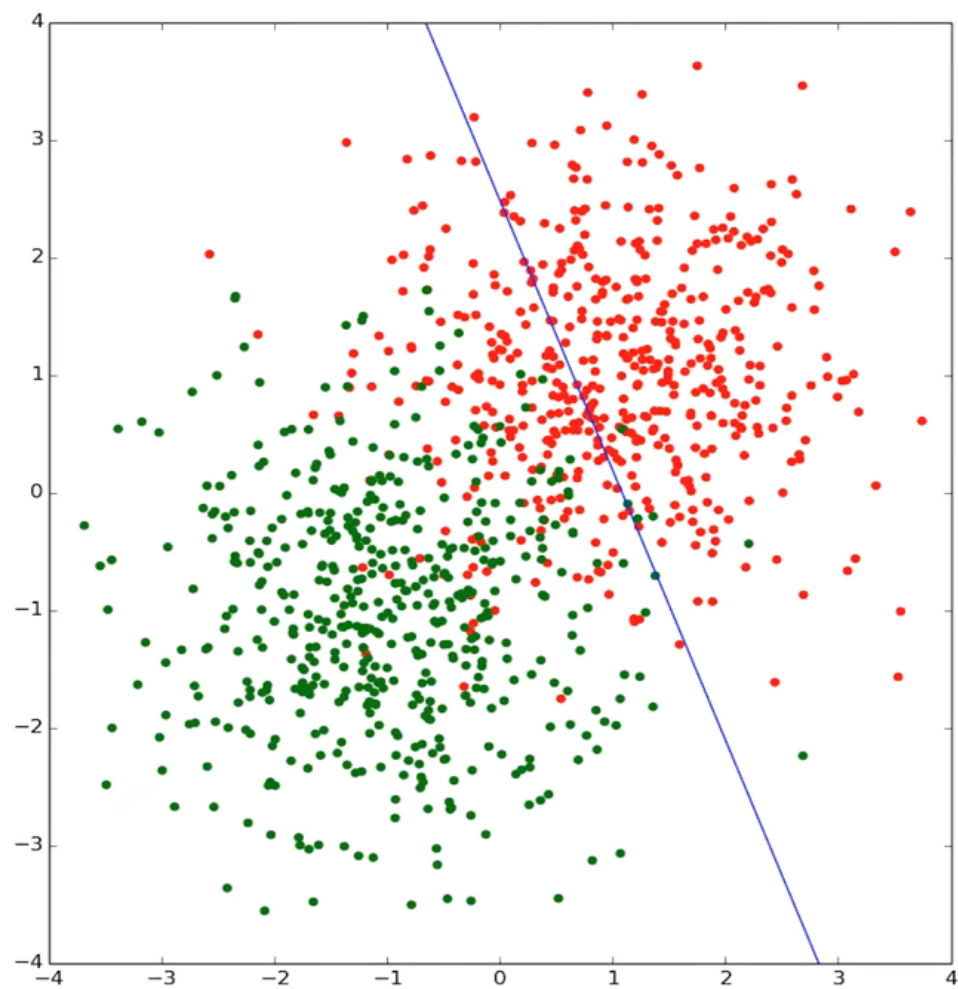
Exemplo de aplicação do Perceptron



Base de dados MNIST



Arquitetura da rede



Treinamento do Perceptron

```
import random, copy

class Perceptron:

    def __init__(self, amostras, saidas, taxa_aprendizado=0.1, epocas=1000, limiar=-1):

        self.amostras = amostras # todas as amostras
        self.saidas = saidas # saídas respectivas de cada amostra
        self.taxa_aprendizado = taxa_aprendizado # taxa de aprendizado (entre 0 e 1)
        self.epocas = epocas # número de épocas
        self.limiar = limiar # limiar
        self.num_amostras = len(amostras) # quantidade de amostras
        self.num_amostra = len(amostras[0]) # quantidade de elementos por amostra
        self.pesos = [] # vetor de pesos
```

Script – 02_Perceptron.py



Fórmula de Atualização dos pesos

A fórmula de atualização dos pesos é dada por:

$$w_j(t + 1) = w_j(t) + \eta \cdot (d - y) \cdot x_j$$

Onde:

- $w_j(t + 1)$ é o peso atualizado após a iteração t .
- $w_j(t)$ é o peso atual na iteração t .
- η é a taxa de aprendizado, um valor que controla o tamanho do ajuste dos pesos (geralmente um valor pequeno entre 0 e 1).
- d é a saída desejada para a amostra atual.
- y é a saída calculada pela rede para a amostra atual.
- x_j é o valor da j -ésima característica da amostra.



Regras de Atualização dos pesos

1. **Inicialização dos Pesos:** Os pesos são inicializados com valores pequenos e aleatórios. Um peso adicional, correspondente ao bias, também é inicializado.
2. **Cálculo da Saída:** Para cada amostra de treinamento, a saída da rede é calculada como uma soma ponderada das entradas mais o bias. A função de ativação (degrau bipolar) é então aplicada a essa soma para determinar a saída binária da rede.
3. **Cálculo do Erro:** O erro para cada amostra é calculado como a diferença entre a saída desejada d e a saída predita y .
4. **Ajuste dos Pesos:** Com base no erro calculado, cada peso w_j é ajustado de acordo com a fórmula de atualização dos pesos. Este ajuste é feito para todas as características da amostra, incluindo o bias.
5. **Iteração:** O processo é repetido para todas as amostras de treinamento e continua por várias iterações (épocas) até que o erro seja minimizado ou o número máximo de épocas seja atingido.

Função de Custo do Perceptron

$$J = \sum_{i=1}^N L(d_i, y_i)$$

J – Custo total

L(d_i,Y_i) – Diferença entre valor predito e valor real

Em geral, o Perceptron usa uma função de erro simples, como o número absoluto de erros de classificação

Conceito de ADALINE

O **ADALINE** (*Adaptive Linear Neuron* ou *Neurônio Linear Adaptativo*) é um modelo de rede neural desenvolvido por Bernard Widrow e Ted Hoff em 1960. Ele é semelhante ao Perceptron, mas utiliza uma função de **ativação linear** em vez de uma **função de degrau**



Principais características do ADALINE

Função de Ativação Linear: A saída é uma combinação linear das entradas ponderadas, sem aplicação de uma função de limiar.

Regra de Aprendizado: Utiliza a regra Delta (ou regra de aprendizado MSE - *min Squared Error*) para minimizar o erro quadrático médio entre a saída desejada e a saída do modelo.

Atualização dos Pesos: Os pesos são ajustados para reduzir o erro de predição, promovendo a convergência do modelo.

Possibilita:

Uso em modelos de **regressão linear**

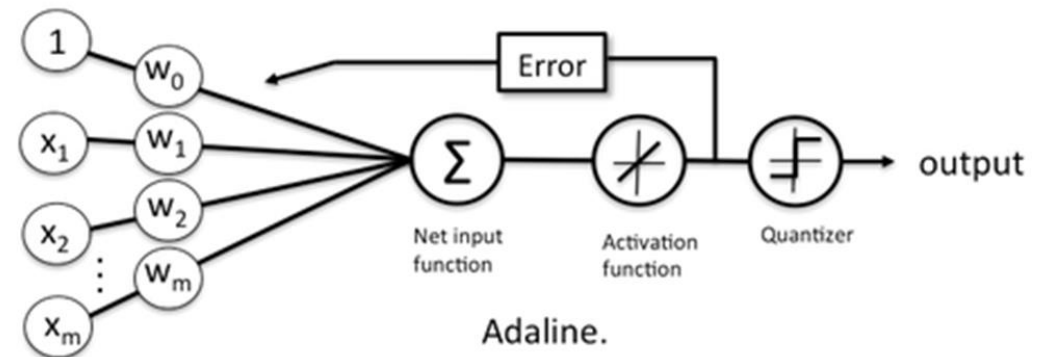
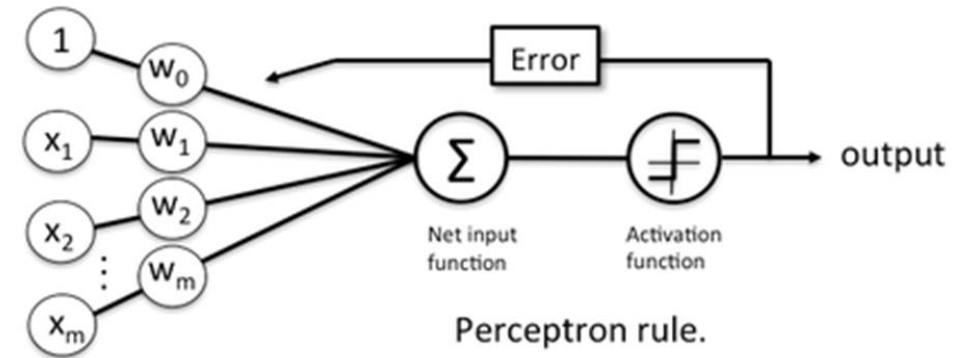
Classificação linear quando usado com um **mecanismo de decisão**.

Estrutura do ADALINE

Regra Delta para atualização dos pesos e minimização do erro

$$\Delta w_j = \eta \cdot (d - y) \cdot x_j$$

Onde η é a taxa de aprendizado.



Regra Delta

A regra Delta, também conhecida como Regra de Aprendizado LMS (*Least Mean Squares*), é um método utilizado para **ajustar os pesos** de uma rede neural de modo a **minimizar o erro** entre a saída desejada e a saída real do modelo

Objetivo

O objetivo da regra Delta é minimizar a função de custo, que geralmente é a soma dos erros quadráticos médios (*Mean Squared Error, MSE*) entre as saídas desejadas e as saídas calculadas.

$$J = \frac{1}{2} \sum_{i=1}^N (d_i - y_i)^2$$

N é o número de amostras.

d_i é a saída desejada para a i -ésima amostra.

y_i é a saída calculada pelo modelo para a i -ésima amostra.



Atualização dos pesos no ADALINE

$$\Delta w_j = \eta \cdot (d - y) \cdot x_j$$

Δw_j é a mudança no peso w_j .

η é a taxa de aprendizado, um parâmetro que controla a magnitude do ajuste dos pesos.

d é a saída desejada.

y é a saída calculada.

x_j é a j -ésima entrada da amostra.

Comparação entre Perceptron e ADALINE

Semelhanças

Modelos de Aprendizado Supervisionado:

Ambos são utilizados para tarefas de classificação supervisionada.

Estrutura:

Ambos utilizam uma soma ponderada das entradas para calcular a saída.

Pesos:

Ambos ajustam os pesos das conexões durante o treinamento.

Comparação entre Perceptron e ADALINE



UNINASSAU

Diferenças

Função de Ativação

- **Perceptron:** Usa uma função de ativação degrau (bipolar) que retorna -1 ou 1.
- **ADALINE:** Usa uma função de ativação linear que retorna um valor contínuo antes de aplicar um limiar de decisão.

Atualização dos Pesos

- **Perceptron:** Atualiza os pesos apenas se a predição estiver incorreta, utilizando a saída da função de ativação.
- **ADALINE:** Atualiza os pesos para todas as amostras, utilizando a saída linear (sem aplicar a função de ativação), minimizando o erro quadrático médio (MSE).

Comparação entre Perceptron e ADALINE

Diferenças

Critério de Convergência

- **Perceptron:** Pode não convergir se os dados não forem linearmente separáveis.
- **ADALINE:** Pode minimizar o erro mesmo que os dados não sejam perfeitamente separáveis linearmente, devido à minimização do erro quadrático médio.

Função de Custo

- **Perceptron:** Não possui uma função de custo explícita.
- **ADALINE:** Usa a função de erro quadrático médio (MSE) para ajustar os pesos.

Recapitulando

O **ADALINE** pode ser visto como um **Perceptron** com algoritmo de treinamento baseado em minimização de um índice de desempenho médio sobre o arquivo de treinamento (algoritmo LMS).

- A especificação dos pesos do ADALINE segue critérios bem definidos que visam minimizar o erro quadrático médio (EQM) na saída do ADALINE.
- O elemento adaptativo do ADALINE é o Combinador Linear Adaptativo (CLA), e a saída do ADALINE pode ser binária, se a função de ativação utilizada for a hardlimiter (degrau), ou contínua, se a função for do tipo sigmóide (ou linear).
- Quando a saída do ADALINE for contínua, o elemento realiza a tarefa de regressão.
- O ADALINE é capaz de solucionar problemas linearmente separáveis não determinísticos (com ruído).

Próximos passos

Introdução a Arquitetura de Redes Neurais

