UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTAÇÃO E SISTEMAS
CAMPUS JOÃO MONLEVADE

Disciplina.....: Inteligência Artificial
Curso.....: Sistemas de Informação
Professor.....: Talles Henrique de Medeiros
Aluno.....: Natanael Emilio da Costa Matrícula: 16.1.8298

Tema: Redes Neurais Perceptron e aplicações usando linguagem Python.

Problema: Reconhecimento de E-mails SPAM usando uma RNA (Rede Neural Artificial) Perceptron.

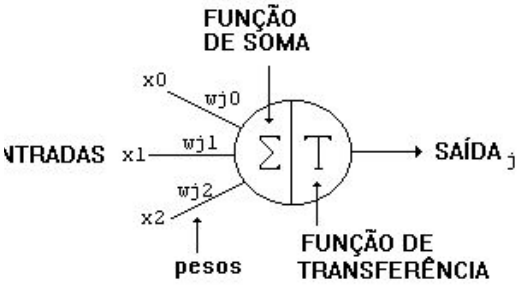
Metodologia: Carregar a base de dados utilizada no exercício prático do KNN para treinar o Perceptron.

O presente documento é um um relatório descrevendo a teoria do perceptron (estrutura do neurônio e algoritmo de treinamento) e o resultado médio obtido ao longo de vários treinamentos.

1 Estrutura do neurônio

Baseado na hipótese do neurônio, alguns dos trabalhos mais antigos de IA tiveram o objetivo de criar redes neurais artificiais, amparado em um modelo matemático simples do neurônio desenvolvido por McCulloch e Pitts (1943). Podemos dizer que o neurônio obedece à lei do "0 ou 1" ou "tudo ou nada" independentemente de como afirmamos, o neurônio sempre estará em um de dois estados: ativado ou desativado (1 ou 0), sendo que as entradas sempre serão recebidas de forma síncrona. Quando uma combinação linear de suas entradas excede algum limiar (rígido ou suave) o neurônio é acionado, ou seja, ele implementa um classificador linear. Sendo assim podemos dizer que uma rede neural nada mais do que apenas uma coleção de unidades conectadas ou seja "neurônios".

Figura 1 - Modelo matemático de um neurônio desenvolvido por McCulloch e Pitts

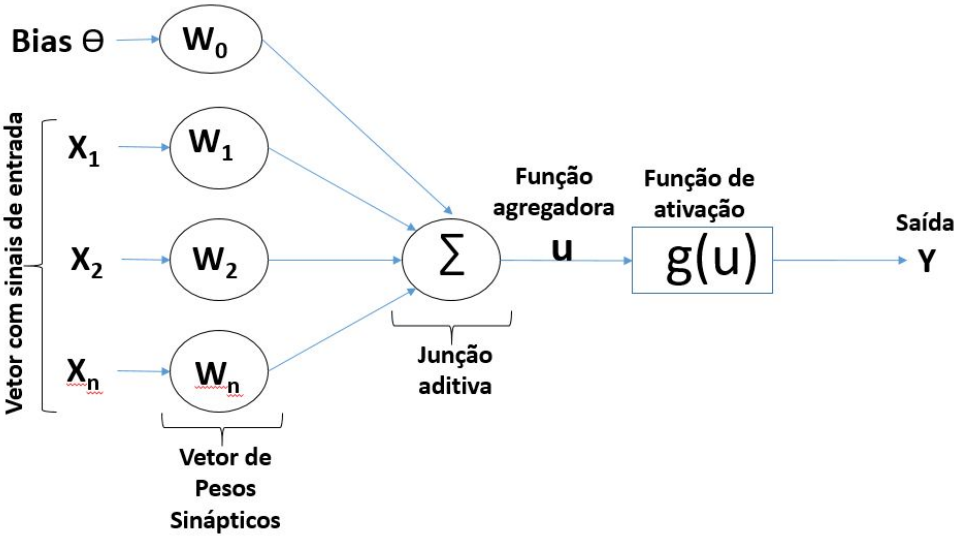


fonte:http://redesneuraisartificiais.blogspot.com/2010/10/o-primeiro-modelo-de-um-neuronio-criado.html

2 Perceptron.

Perceptron é uma rede neural de camada única e um Perceptron de várias camadas é chamado de Rede Neural Artificial. O Perceptron é um classificador linear (binário). Mas antes é importante entender que as redes neurais são compostas por unidades(neurônios) conectadas por ligações direcionadas. Uma ligação da unidade "a" para a unidade "b" serve para propagar a ativação e para cada ligação também tem um peso numérico associado a ele, que determina a força e o sinal de conexão.

Figura 2 - Representação do modelo do perceptron



https://www.linkedin.com/pulse/redes-neurais-artificiais-john-costa

Escolher os pesos adequados pode ser simples para a representação das operações booleanas, mas deixa de ser quando se quer fazer reconhecimento de padrões mais complexos e com base nesse problema que outros pesquisadores desenvolveram técnicas de ajustamento de pesos com o intuito de fazer com que os neurônios aprendessem o padrão de entrada e assim o Modelo Perceptron foi desenvolvido nas décadas de 1950 e 1960 pelo cientista Frank Rosenblatt, baseando nos trabalhos anteriores de Warren McCulloch e Walter Pitts.

Sendo uma rede com todas as entradas conectadas diretamente com as saídas é chamada de rede neural de camada única ou rede Perceptron.

1.3 Algoritmo de treinamento

Para que uma rede dessas funcione, é preciso treiná-la e o algoritmo responsável pelo aprendizado da rede perceptron é o backpropagation, o processo de treinamento de um modelo Perceptron consiste em fazer com que o modelo aprenda os valores ideais de pesos e bias. O algoritmo de treinamento pode ser separado em dois grupos:

<u>Supervisionado:</u>

- O treinamento é realizado utilizado pares entrada - saída alvo;
- Os pesos sinápticos são ajustados visando minimizar o erro entre a saída alvo e a saída real da rede.

Não-supervisionado:

- Não existe saída alvo para o problema;
- O treinamento busca obter uma nova representação dos padrões de entrada segundo algum critério pré estabelecido.

Para o treinamento é indicado que o conjunto de dados disponível seja dividido em três grupos:

- Conjunto de treino - utilizado para o treinamento propriamente dito, sou seja para realizar o aprendizado;
- Conjunto de validação - utilizado para verificar, durante o treinamento como evolui o erro da rede para sinais desconhecidos;
- Conjunto de teste - utilizado para verificar o desempenho da rede para amostras desconhecidas.

No caso do Perceptron nós apresentamos ao modelo os dados de entrada e as possíveis saídas, treinamos o modelo e pesos e bias são aprendidos.

Com o modelo treinado, podemos apresentar novos dados de entrada e o modelo será capaz de prever a saída e assim com base no cálculo do erro ocorrido na camada de saída da rede neural, recalcular o valor dos pesos do vetor w da camada última camada de neurônios e assim proceder para as camadas anteriores, de modo que possa atualizar todos os pesos w das camadas a partir da última até atingir a camada de entrada da rede, para isso realizando a retropropagação.

2. O que foi feito

2.1 Instalações e importações

Para a leitura, separação e manipulação dos dados foram utilizadas as bibliotecas:

- from csv import reader
- import pandas as pd
- import random
- import numpy as np

Para plotar o gráfico Erro x Tempo e gera a matriz de confusão precisa ter instalado as bibliotecas abaixo:

- py -m pip install -U pip
- py -m pip install -U scikit-learn
- py -m pip install -U matplotlib

A biblioteca scikit-learn é utilizada para gerar a matriz de confusão, e a matplotlib fornece os recursos para gera os gráficos necessários. Assim sendo utilizadas com a importar das biblioteca:

- import pylab as plotagem
- from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
- from sklearn.model_selection import train_test_split

2.1 Leitura e separação dos dados

A base de dados foi dividida em duas: uma para treinamento e uma para teste, na proporção de 60% e 40%. Os dados foram escolhidos aleatoriamente para não gerar conjuntos de dados viciados ou polarizados.

Os dados foram lidos como *data frames* e transformados em list para melhor utilização, aproveitando recurso de ambas as classes.

2.2 Modificando o algoritmo de aprendizagem

Para treinar o meu Perceptron com o conjunto de dados de treinamento foi preciso adicionar vetores auxiliares e um critério de parada pois a base de dados possui uma sobreposição espacial dos dados e por isso, o Perceptron não conseguirá separar perfeitamente os dados por meio de uma superfície linear de separação.

Fora adicionadas as listas:

- listaDeErros = lista de quantidade de Erros na etapa de treinamento
- listaDeEpocas = lista para armazenar as épocas
- valoresObtidos = lista respostas obtidas na época - usado para calcular o número de erros

Com a operação - sum((1) for a, b in zip(valoresObtidos, self.exit) if a!=b) - pode-se calcular a quantidade de erros que ocorreram na época.

Para ajudar a decidir o ponto de parada seleciono os 999 penúltimos números da lista de erros - listaDeErros[-1000 : -1] - para que possa comparar com o último e verificar se valor do erro está com tendência a aumentar ou estagnar. Com o auxílio da operação - [numero for numero in nUltimos if numero < listaDeErros[-1]] - verifica se o último valor de erro é maior que todos os 999 anteriores a ele, caso o seja está apresentando indícios de não reduzir ou até mesmo começar a aumentar, caso seja confirmado antes do erro chegar a 0 o algoritmo para o treinamento.

Para montar o gráfico do treinamento Erros x Tempo(épocas), o algoritmo retorna uma tupla - return(listaDeErros, listaDeEpocas) - contendo o vetor com todas as épocas e a quantidade de erros encontrados em cada uma delas.

A função - def sort(self, sample) - passa a retornar os valores que identificar se o elemento avaliado é non-spam ou spam (0 ou 1), facilitando criar um vetor contendo todas as respostas do teste para gerar a matriz de confusão.

2.3 Valores encontrados para as matrizes de confusão

Para testar e avaliar a performance do algoritmo modificado foi utilizada a matriz de confusão das medidas de sensibilidade e especificidade.

Dentro deste contexto foram realizados 10 testes contendo 40 % das entradas do csv, selecionados de forma aleatória para garantir que o conjunto de testes não esteja viciado o tendencioso. Assim podemos obter a tabela abaixo.

Os gráficos de Erro x Tempo que se destacaram foram inseridos após.

Tabela 1 - Tabulação das matrizes de confusão geradas nos testes

Nº do teste	Qtd. Épocas	Sensibilidade	Acurácia	Especificidade	Matriz Confusão [[linha 1] [linha 2]]
01	**7193**	**0.9163**	**0.9255**	**0.9400**	**[[1029 94] [43 674]]**
02	**6216**	**0.9323**	**0.9326**	**0.9331**	**[[1047 76] [48 669]]**
03	**6589**	**0.8219**	**0.8815**	**0.9749**	**[[923 200] [18 699]]**
04	**7158**	**0.9314**	**0.9315**	**0.9317**	**[[1046 77] [49 668]]**
05	**8083**	**0.9341**	**0.9337**	**0.9331**	**[[1049 74] [48 669]]**
06	**6412**	**0.9314**	**0.9315**	**0.9317**	**[[1046 77] [49 668]]**
07	**6660**	**0.8219**	**0.8815**	**0.9749**	**[[923 200] [18 699]]**
08	**7878**	**0.9341**	**0.9337**	**0.9331**	**[[1049 74] [48 669]]**
09	**7303**	**0.9207**	**0.9283**	**0.9400**	**[[1034 89] [43 674]]**
10	**6768**	**0.9323**	**0.9321**	**0.9317**	**[[1047 76] [49 668]]**
Média	**7026**	**0.9076**	**0.9212**	**0.9424**	**-**

.

Figura 3: Gráfico Erros x Tempo - Teste 1

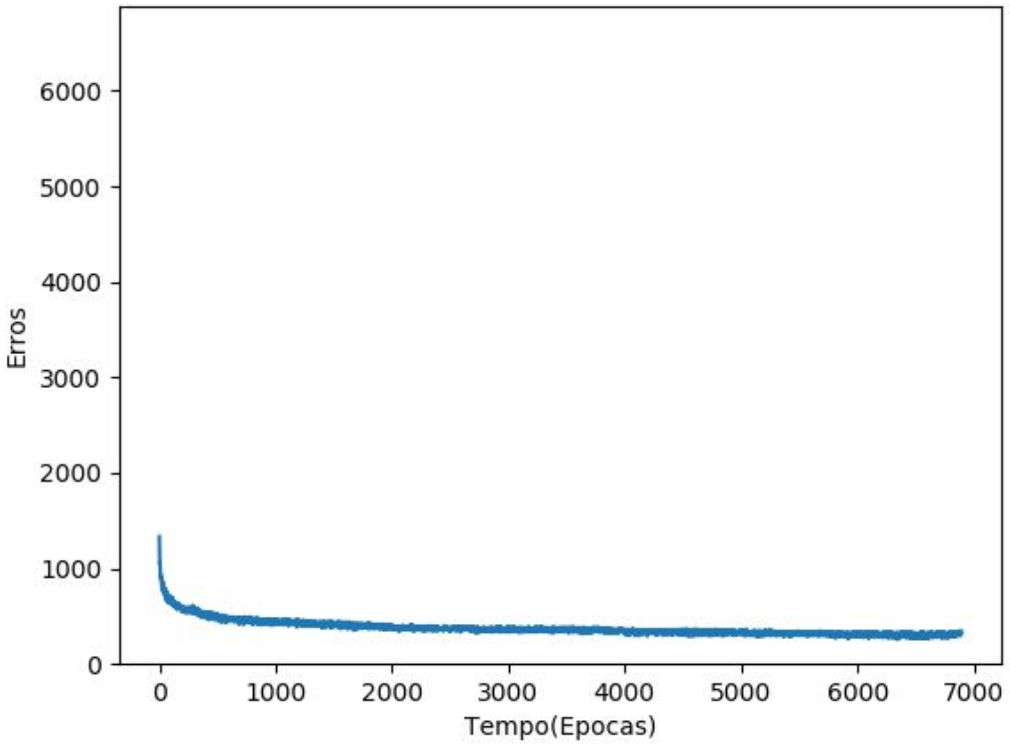


Figura 4: Gráfico Erros x Tempo - Teste 2

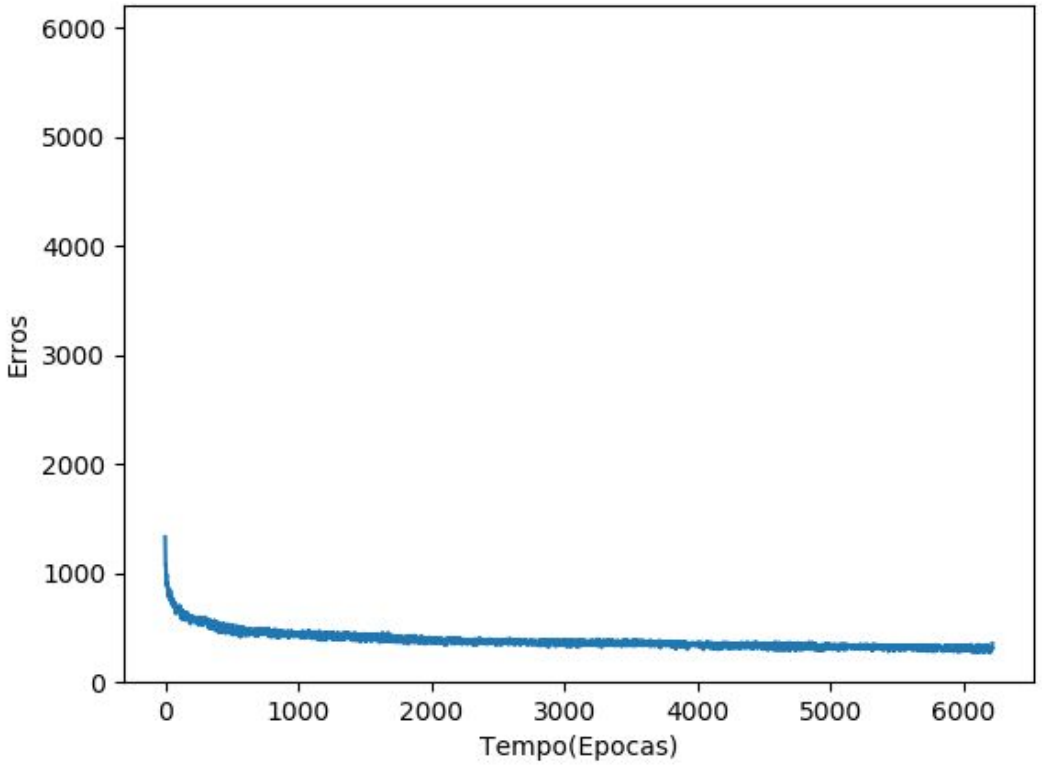
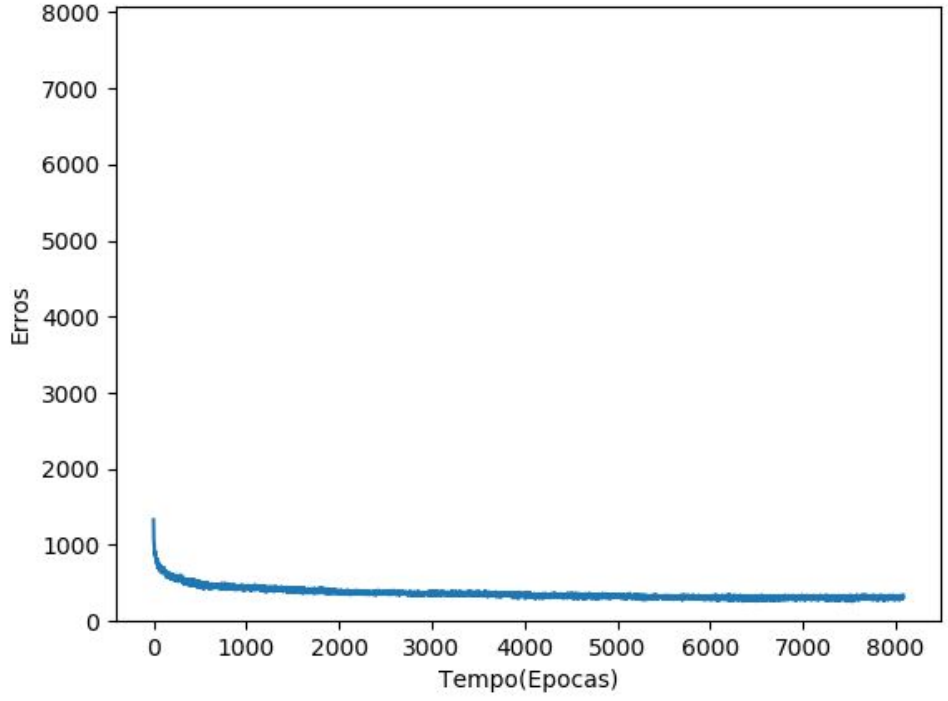


Figura 5: Gráfico Erros x Tempo - Teste 5



3. Conclusão

O perceptron simples é um algoritmo de aprendizagem de máquinas, facilmente implementado para casos mais simples ou seja objetos com poucos números de variáveis, para casos mais complexos são usadas estruturas com mais neurônios, ou que utilizem outros algoritmos de treinamento.

O importante é entender a complexidade do problema que se propõe a resolver e escolher o algoritmo que melhor se adequa à necessidade.

4. Referências

Russell, S. & Norvig, Peter. Inteligência artificial. Rio de Janeiro: Campus, 2004.