# Relatório do Trabalho III
# Inteligência Artificial

Alunos: Edson Lemes da Silva e Lucas Cezar Parnoff

17 de Julho de 2017

## 1 Descrição geral

Esta aplicação permite fazer a leitura de um conjunto específico de dados, e então separá-los em um conjunto de treino e outro de teste.

A rede neural utilizada neste projeto usa um método de aprendizagem sem supervisão, isto é, não há acompanhamento do conjunto de saída durante o processo de treinamento da rede. O algoritmo utilizado, é o de Kohonen. Este algoritmo procura agrupar os padrões de dados, desta maneira consegue distinguir exemplos de diferentes classes.

Assim, esta aplicação permite ler, treinar e classificar exemplos de acordo com uma rede já treinada. O treinamento para alguns conjuntos são realmente lentos, tais como para o conjunto utilizado neste projeto. Assim, a aplicação permite salvar e carregar uma rede para ser usada em algum momento.

A linguagem de programação utilizada neste projeto, é o Python. Para fins de compatibilidade, foi considerado a execução apenas para a segunda versão do Python, isto é, atualmente na versão 2.7.

## 2 Conjunto de dados

O conjunto utilizado no trabalho esta nomeado como: "optdigits.data", onde apresenta 1934 exemplos de dígitos manuscrito, cada um deles descritos em uma matriz com dimensões de 32x32. Para simplificar a leitura deste conjunto, foi eliminado o cabeçalho deste arquivo original. Assim, há apenas os dados e seus respectivos rótulos (respostas) no conjunto de entrada.

A leitura deste conjunto ocorre com o auxílio das funções definidas no arquivo "readfile.py". A primeira função chamada de `readFile` recebe um parâmetro que é o nome do arquivo de entrada. A segunda função chama-se `divideSet`, ela simplesmente divide o conjunto lido em duas partes : treino e teste com 70% e 30% dos dados, respectivamente.

Utilizando o pacote `Numpy`, cada exemplo lido no arquivo original com dimensões de 32x32, é convertido para um vetor de 1024 posições somado com o rótulo associado, isto é, 1025 posições. Assim, o arquivo lido será uma instância de uma matriz do tipo `numpy`;

o seu tamanho será de 1934x1025, ou seja, todos os exemplos de entrada. A divisão entre conjunto de teste e treino separa os dados dos rótulos, ou seja, será criado para um deles uma matriz com 1024 colunas, e um vetor de rótulos com uma coluna.

```
from readfile import *

conj = readFile("optdigits.data")

#Xtr,ytr para o treino e Xte,yte para teste
Xtr,ytr,Xte,yte = divideSet(conj)
```

Exemplo 1: Carregamento do conjunto

## 3 O algoritmo de Kohonen

A biblioteca utilizada para o algoritmo de Kohonen esta implementada em Python, a sua versão original encontra-se em um repositório na Internet. Esta implementação foi modificada para atender as necessidades do projeto.

Nesta biblioteca estão implementadas duas classes, uma representa um mapa auto-organizável e a outra um neurônio. Ambos são nomeados como `class Map` e `class Neuron` respectivamente. Essas implementações estão no arquivo "som.py".

O construtor da classe `Map` recebe como parâmetro a quantidade de atributos (dimensão), número de neurônios (considere que trabalha apenas com matriz quadrada), conjunto de treino e teste (para ambos deve-se passar também o conjunto de rótulos respectivos). A seguir será descrito algumas funções principais desta classe.

A primeira função principal a ser descrita é em relação ao treinamento de uma rede. A função `initWeights` inicializa os neurônios e também seus pesos com valores aleatórios entre 0 e 1. A quantidade de neurônio é definida em forma de matriz quadrada( e.g. para o valor 10, será gerado uma matriz 10x10, ou seja, 100 neurônios).

A segunda função principal a ser destacada é a de treinamento, nomeada como `train` dentro da classe `map`. Esta recebe como parâmetro um valor de erro, que representa a condição de parada do algoritmo, e também um valor booleano que define a impressão ou não dos testes internos durante o treinamento. Deste modo, escolhe-se dentro do conjunto de treino um exemplo e calcula o neurônio vencedor a partir dele, e também faz a atualização dos pesos conforme a influência e raio da vizinhança. O erro retornado é em relação as atualizações dos pesos, isto é, a média obtida através da soma dos pesos atualizados.

A próxima função chama-se `trainPattern`, ela recebe um exemplo como parâmetro e faz a chamada para as demais funções de atualização. Primeiramente ela obtém o neurônio vencedor em relação ao exemplo testado, posteriormente chama a função de atualização de pesos. Nesta etapa também é feito a associação do rótulo ao neurônio vencedor.

A função que verifica o neurônio vencedor chama-se `winner`, ela percorre todos os neurônios e computa a distância euclidiana quadrada para seus respectivos pesos em relação ao exemplo passado como parâmetro.

Em relação ao carregamento e salvamento de redes neurais, elas podem ser salvas em um arquivo externo através da função `saveNetwork` passando como parâmetro o nome do arquivo desejado. Para fazer o carregamento, primeiramente utiliza a função loadWeights do arquivo "readfile.py" passando como parâmetro o nome do arquivo desejado para fazer o carregamento. Posteriormente, usa-se a função `loadNetwork` da classe `Map` para carregar os pesos de cada neurônio.

Agora serão descritos as funções referentes a classe `Neuron`. A principal delas chama-se `updateWeights`, ela realiza a atualização dos pesos em relação ao neurônio vencedor e ao exemplo sendo analisado. A partir dela que são chamadas as demais funções, tais como as que calculam a influência da vizinhança e decaimento da taxa de aprendizagem.

A função `learningRate` calcula a taxa de aprendizagem, inicialmente definida como 0.1; e acontece a atualização desta taxa por esta função, com o intuito de diminuí-la.

Outra parte do processo de atualização de pesos é a influência da vizinhança, a topologia dela. Esse calculo ocorre através da função gaussiana descrito na implementação por `gauss`, implementada dentro da classe `Neuron`.

E a última função é a que define o raio de influência (tamanho), descrito na implementação como `strength`, o seu cálculo leva em conta a iteração atual e o valor de sigma inicial.

A classificação da rede ocorre através da função `classify` ela chama outros métodos que a partir do conjunto de teste, verifica para cada neurônio a distância euclidiana, assim aquele que tiver a menor distância atribui-se um rótulo para aquele neurônio conforme o exemplo sendo verificado. No final, é plotado na tela os neurônios com seus respectivos rótulos, cada um com uma cor correspondente. A função recebe como parâmetro um vetor que são metadados sobre a rede sendo carregada( caso a rede já estiver carregado na memória, a função não necessita do parâmetro).

# 4 Execução

Para a execução desta aplicação, leva-se em conta o Python 2 (como forma de compatibilidade). A execução pode ser feita de duas formas, através do Makefile, que seleciona um exemplo e executa o Python; e a outra forma é diretamente do interpretador Python, que pode ser aberto em um terminal digitando "python", desta maneira, é precisa especificar o arquivo a ser interpretado utilizando o comando "execfile(NOME_ARQUIVO) ".

Inicialmente carrega-se o conjunto de dados, utilizando as funções do arquivo "readfile.py', então cria-se uma instância da classe `Map` passando os parâmetros para o construtor. Assim, há duas opções ou aplicar o algoritmo de treino sobre a rede, ou carregar uma rede já treinada para análise de resultados. Para trenar chama-se o método `train` passando como parâmetro o erro para condição de parada e a flag booleana para debug. Para carregar uma rede, chama-se a função `loadNetwork` passando como parâmetro a matriz de pesos carregada do arquivo e o vetor de rótulos para rede. A partir daí, é possível chamar a função `classify` que irá avaliar a rede e exibir o erro em relação as respostas corretas.

Algumas dependências (pacotes) são necessárias para a execução correta, posteriormente

serão descritos as instruções para instalação. No diretório deste projeto estão disponíveis dois exemplos, um para treinamento da rede e outro para carregamento e classificação, chamados de “treina.py” e “testa.py” respectivamente.

```
import random
import math
import numpy as np
import matplotlib.pyplot as plt
from readfile import *

#Carrega os arquivos com readfile
#parametros(dim,qtd neuronios,conj. treino,conj. teste, rotulos de teste)
n = Map(Xtr.shape[0],10,Xtr,ytr,Xte,yte)

#Erro desejado e flag de debug
n.train(0.0001,True)

#Classifica a rede
n.classify(None)
#Salva a rede em um arquivo externo, passando nome como param
n.saveNetwork("rtreinada10.data")
```

Exemplo 2: Treinamento de uma rede

```
import random
import math
import numpy as np
import matplotlib.pyplot as plt
from readfile import *


#carrega os arquivos com readfile

#carrega os pesos de uma rede passando nome do arquivo
weights,labels,header = loadWeights("rtreinada.data")
#parametros(dim,qtd neuronios,conj. treino,conj. teste, rotulos de teste)
n = Map(Xtr.shape[0],10,Xtr,ytr,Xte,yte)
n.loadNetwork(weights,labels)

#classifica passando o conj. de teste e seus rotulos
n.classify(header)
```

Exemplo 3: Carregamento de uma rede

## 5 Resultados

Durante o treinamento, o algoritmo mostra algumas situações sobre a iteração ocorrendo, tais como: exemplo selecionado e o seu rótulo, o neurônio vencedor e no fim exibe uma

mensagem de confirmação após a atualização dos pesos em relação a este exemplo. Para não ficar muito poluído a visualização, permite-se imprimir ou não estes dados sobre o treinamento, e para isto, é passada como parâmetro um valor booleano para a função `train`; sendo que verdadeiro significa imprimir as ações durante a impressão, e falso o contrário. Após a verificação de todo o conjunto de treino, é exibido na tela o erro atual em relação aos pesos atualizados.

Os resultados da rede após o treinamento, pode ser verificado através da função `classify`, ela irá analisar o conjunto de teste e classificar a capacidade de cada neurônio em reconhecer este conjunto. Assim, após esta computação, a aplicação irá plotar na tela um gráfico contendo os neurônios com seus respectivos rótulos associados, sendo que cada classe terá uma cor diferente para representar cada neurônio (cada ponto no plano cartesiano).

Como exemplo de resultados, no diretório do projeto há duas redes treinadas, elas estão nos arquivos "rtreinada.data" e "rtreinada1.data"; a primeira há 36 neurônios, com erro de 0.45 em relação a atualização de pesos, e o segundo exemplo há 16 neurônios com erro de $10^{-6}$. É importante destacar que estes dois exemplos foram obtidos com um tempo razoável de computação, assim o treinamento deste algoritmo depende do erro da condição de parada.

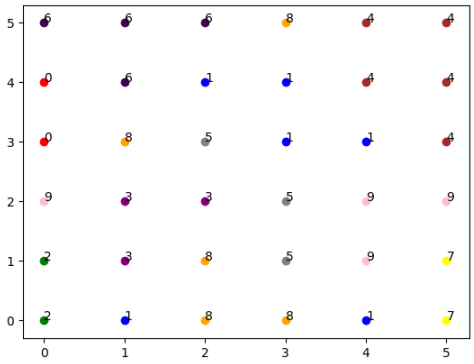


Figura 1: Rede de Kohonen com 36 neurônios e erro de 0.45

A Figura 1 representa uma rede com 36 neurônios, com erro de 0.45. Esta classificação foi obtida pela análise do conjunto de testes. Deste modo, para cada neurônio verificou-se a distância para todos os exemplos do conjunto de teste, assim aquele que corresponde a menor distância euclidiana quadrada, o algoritmo compara o rótulo daquele exemplo para o neurônio sendo testado. A classificação em relação ao conjunto de teste apresenta uma acurácia de 69.45 %.

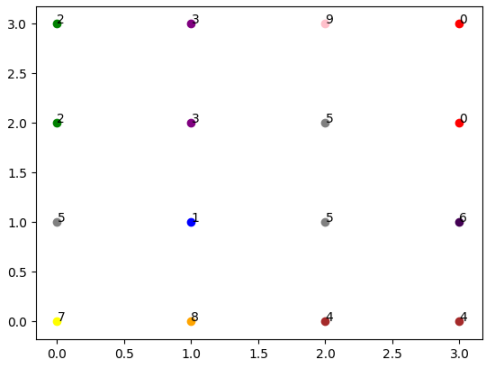


Figura 2: Rede de Kohonen com 16 neurônios e erro de $10^{-6}$.

E por último, a Figura 2 mostra uma rede treinada com 16 neurônios com erro de $10^{-6}$. Assim, minimamente consegue-se classificar as classes com uma quantidade reduzida de neurônios. Neste caso, a acurácia obtida foi de 81.25%.

As duas redes mostradas acima estão salvas nos arquivos "rtreinada.data", "rtreinada1.data" respectivamente. Elas podem ser carregadas passando como parâmetros para as funções de leitura, conforme exemplificado no arquivo "testa.py". Para cada um delas é gerado um arquivo com mesmo nome concatenado com o prefixo "l", representando os rótulos aplicados a cada neurônio.

## 6 Instalação de pacotes

Para execução deste algoritmo, alguns pacotes são necessários, entre eles são o `Numpy` e `matplotlib`. O primeiro é usado na leitura do conjunto de entrada e também durante o processo de treinamento. O segundo serve para plotar gráficos, para facilitar a visualização da rede em si.

Deste modo, assumindo um ambiente Linux e o Python 2.7 como padrão. Para instalação, usando o terminal basta digitar:

- **Numpy:** pip install numpy
- **Matplotlib:** sudo apt-get install python-matplotlib

De um modo geral, somente estes pacotes são necessários instalar, já que os demais são nativos do Python.

## 7 Links

- Biblioteca Kohonen, por Marcel Caraciolo : http://aimotion.blogspot.com.br/2009/04/redes-neurais-auto-organizaveis-som.html

- Numpy : http://www.numpy.org/
- Matplotlib : https://matplotlib.org/