**Atividade 5** **Yan A. S. Duarte**
PEL 208 Tópicos Especiais em Aprendizagem
Prof. Reinaldo A. C. Bianchi Entrega: 22/11/2017

# Introdução

Esse relatório tem como objetivo detalhar a teoria, a implementação, os resultados e a conclusão da quinta atividade do curso. A propósta do exercício é implementar uma rede neural artificial utilizando back-propagation.

# Teoria

## Rede Neural Artificial

As Redes Neurais Artificiais (RNA) são técnicas que se inspiram na estrutura neural biologica e podem adquirir conhecimento a partir de experiências.

Uma RNA possui unidades de processamento que são conectadas por canais associadas a um determinado peso. Essas unidades executam operações em cima de seus dados de entrada gerando assim um dado de saída.

Podemos descrever o funcionamento de uma unidade de processamento da seguinte forma:

- Sinais são apresentados à entrada;
- Cada sinal é multiplicado por um peso, que indica a sua influência na saída da unidade;
- Soma-se os sinais produzindo um nível de atividade;
- Se este nível de atividade ultrapassar um limiar a unidade produz uma determinada saída.

Também é possível ajustar os pesos das conexões das RNAs de acordo com uma regra de treinamento, isso faz com que a rede aprenda a partir de exemplos.

A arquitetura de uma rede neural é organizada em camadas que são conectadas a outras camadas. Podemos classificar essas camadas em tres diferentes grupos::

- Camada de entrada: onde os valores iniciais são inseridos na rede;
- Camada escondida: onde é feito o processamento;
- Camada de saída: onde se obtem o resultado final.

### Aprendizado

Uma característica importante da RNA é a possibilidade de aprendizado, melhorando assim o seu desempenho. Para isso, um processo iterativo de treinamento realiza o ajuste de cada peso. O aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas.

Para um problema de aprendizado supervisionado, onde se sabe a resposta desejada para o padrão de entrada, podemos corrigir os pesos de duas maneiras:

- Modo Padrão: Corrigi os pesos a cada amostra de entrada do conjunto de treinamento. Essa correção baseia-se no erro do exemplo apresentado naquela iteração. Assim, em cada ciclo ocorrem N correções.

- Modo Batch: Apenas uma correção é feita por ciclo. Todos os exemplos do conjunto de treinamento são apresentados à rede, seu erro médio é calculado e a partir deste erro fazem-se as correções dos pesos.

Quando um padrão é inicialmente apresentado à rede, ela produz uma saída. Após medir a distância entre a resposta atual e a desejada, são realizados os ajustes apropriados nos pesos das conexões de modo a reduzir esta distância. Este procedimento é conhecido como Regra Delta.

## Implementação

Para a elaboração do exercício foi utilizada a linguagem de programação python. Foram criadas funções separadas para cada etapa da rede neural.

### Função inicia_rede

Função que retorna uma rede iniciada de acordo com os parâmetros enviados.

```
def inicia_rede(num_entradas, num_escondidas,
        num_saidas):
    rede = list()
    camada_escondida =
        [{'pesos':[random() for i in range(num_entradas + 1)]}
            for i in range(num_escondidas)]
    rede.append(camada_escondida)
    camada_saida =
        [{'pesos':[random() for i in range(num_escondidas + 1)]}
            for i in range(num_saidas)]
    rede.append(camada_saida)
    return rede
```

### Função forward_propagate

Função para calcular o foward propagate de uma rede.

```
def forward_propagate(rede, linha):
    entrada = linha
    for camada in rede:
        nova_entrada = []
        for neuronio in camada:
            ativacao = ativar(neuronio['pesos'], entrada)
            neuronio['saida'] = transferir(ativacao)
            nova_entrada.append(neuronio['saida'])
        entrada = nova_entrada
    return entrada


# Calcula a ativacao do neuronio para uma entrada
def ativar(pesos, entradas):
    ativacao = pesos[-1]
    for i in range(len(pesos)-1):
```

```
        ativacao += pesos[i] * entradas[i]
    return ativacao


# Transferir a ativacao do neuronio
def transferir(ativacao):
    return 1.0 / (1.0 + math.exp(-ativacao))


# Calcula a derivada de uma saida de neuronio
def transferir_derivada(saida):
    return saida * (1.0 - saida)
```

### Função backward_propagate_erro

Função para calcular o erro do backward propagate.

```
def backward_propagate_erro(rede, esperado):
    for i in reversed(range(len(rede))):
        camada = rede[i]
        erros = list()
        if i != len(rede)-1:
            for j in range(len(camada)):
                erro = 0.0
                for neuronio in rede[i + 1]:
                    erro += (neuronio['pesos'][j] *
                            neuronio['delta'])
                erros.append(erro)
        else:
            for j in range(len(camada)):
                neuronio = camada[j]
                erros.append(esperado[j] - neuronio['saida'])
        for j in range(len(camada)):
            neuronio = camada[j]
            neuronio['delta'] = erros[j] *
                transferir_derivada(neuronio['saida'])
```

### Função treinar_rede

Função para treinar uma rede de acordo com os parâmetros enviados.

```
# Treine uma rede para um determinado numero de epocas
def treinar_rede(rede, treino, taxa_aprendizado,
        num_epocas, num_saidas):
    for epoca in range(num_epocas):
        soma_erro = 0
        for linha in treino:
            saidas = forward_propagate(rede, linha)
            esperado = [0 for i in range(num_saidas)]
            esperado[linha[-1]] = 1
```

```
            soma_erro += sum([(esperado[i]-saidas[i])**2
                              for i in range(len(esperado))])

            backward_propagate_erro(rede, esperado)
            atualiza_pesos(rede, linha, taxa_aprendizado)

# Atualiza pesos da rede com devido erro
def atualiza_pesos(rede, linha, taxa_aprendizado):
    for i in range(len(rede)):
        entradas = linha[:-1]

        if i != 0:
            entradas = [neuronio['saida']
                        for neuronio in rede[i - 1]]

        for neuronio in rede[i]:
            for j in range(len(entradas)):
                neuronio['pesos'][j] += taxa_aprendizado *
                    neuronio['delta'] * entradas[j]

        neuronio['pesos'][-1] += taxa_aprendizado *
            neuronio['delta']
```

### Função back_propagation

Função back-propagation

```
# Backpropagation
def back_propagation(treinamento, teste, taxa_treinamento,
        num_epoca, num_escondidas):
    num_entradas = len(treinamento[0]) - 1
    num_saidas = len(set([row[-1] for row in treinamento]))

    rede = inicia_rede(num_entradas, num_escondidas,
           num_saidas)

    treinar_rede(rede, treinamento, taxa_treinamento,
        num_epoca, num_saidas)

    predicoes = list()
    for row in teste:
        pred = predicao(rede, row)
        predicoes.append(pred)
    return predicoes

# Faz uma predicao com uma rede
def predicao(rede, linha):
    saidas = forward_propagate(rede, linha)
```

```
    return saidas.index(max(saidas))
```

Os códigos completos podem ser vistos nos arquivos redeneural.py e testes.ipynb que se encontram na pasta da atividade.

### Testes

Para testar o funcionamento da rede neural artificial foi utilizado o banco de dados de digitos escritos a mão dos códigos postais dos envelopes de cartas dos Estados Unidos.

Cada imagem do banco representa um único digito extraído do código postal. As imagens possuem dimenssão de 16x16 bits e estão em escala de cinza. Nesse trabalho, foram feitos três testes alterando as variáveis da rede neural artificial.

## Resultados

### 1º Teste

Digitos utilizados: 0 e 1
Taxa de Aprendizado: 0.5
Nº de épocas: 5
Nº neurônios na camada escondida: 30
Acuracia: 98.876%

### 2º Teste

Digitos utilizados: 0 e 1
Taxa de Aprendizado: 0.3
Nº de épocas: 10
Nº neurônios na camada escondida: 15
Acuracia: 99.197%

### 3º Teste

Digitos utilizados: 0 e 1
Taxa de Aprendizado: 0.7
Nº de épocas: 3
Nº neurônios na camada escondida: 10
Acuracia: 98.716%

## Conclusão

Nesse relatório foi implementado, em linguagem python, um algoritmo para criar uma rede neural artificial com backpropagation. Para testar o algoritimo, foi utilizado o dataset com imagens de digitos obtidos de códigos postais em correspondências do Estados Unidos.

Podemos observar que, ao aumentar o número de épocas que se realiza a correção do back-propagation, conseguimos um melhor resultado.

## Referências


[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2001.*

*[2] S. Russell, and P. Norvig.* Artificial Intelligence: A Modern Approach. Series in Artificial Intelligence Prentice Hall, Upper Saddle River, NJ, terceira edition, 2010

[3] *Rede neural artificial Disponível em (https://pt.wikipedia.org/wiki/Rede_neural_artificial). Acesso em: 20 de out. de 2017*