Relatório Reconhecimento de Padrões: Atividade 06 − Redes Neurais

Carlos Emmanuel Pereira Alves
Curso de Bacharelado em Ciência da Computação
Universidade Federal do Agreste de Pernambuco (UFAPE)
Garanhuns, Brasil
carlos.emmanuel.236@gmail.com

1) Utilizado a base Iris, https://archive.ics.uci.edu/ml/datasets/Iris. Os 25 primeiros exemplos de cada classe serão o conjunto de treino, os outros exemplos são o conjunto de teste. Faça sua própria implementação de Neurônio Artificial.

Primeiro comecei fazendo a implementação do neurônio, utilizei a função de ativação degrau.

```python
# Função de ativação do neurônio, que no caso será a função degrau
def step(x):
    return np.where(x >= 0, 1, 0)


class Neuron:
    def __init__(self, input_size):
        self.w = np.random.randn(input_size)
        self.b = np.random.randn()

    def forward(self, x):
        z = np.dot(x, self.w) + self.b
        a = step(z)
        return a

    def backward(self, x, y, a, lr):
        d = y - a
        dw = lr * d * x
        db = lr * d
        self.w += dw
        self.b += db
```

a) Treine um neurônio para classificar a classe Iris-setosa como 1 e as demais como zero. Pare o algorítimo de treinamento com erro zero no conjunto de treino. Calcule o erro no conjunto de teste.
Carreguei a base usando a função load_iris, do sklearn.datasets. Também já transformei as classes definindo a iris-setosa como 1 e as demais como 0. Separei então os conjuntos de treino e teste, com os 25 primeiros exemplos de cada classe como o conjunto de treino.

```
iris = load_iris()
X = iris.data
y = iris.target

# Define a Iris-setosa = 1, demais = 0
y[y != 0] = -1
y[y == 0] = 1
y[y != 1] = 0

X_train = np.concatenate((X[:25, :], X[50:75, :], X[100:125, :]))
y_train = np.concatenate((y[:25], y[50:75], y[100:125]))

X_test = np.concatenate((X[25:50, :], X[75:100, :], X[125:150, :]))
y_test = np.concatenate((y[25:50], y[75:100], y[125:150]))
```

Depois fiz a normalização dos dados. Então criei o neurônio e defini o número de épocas e a taxa de aprendizado, depois treinei o neurônio, me atentando para quando a taxa de erro for 0 o algoritmo parar. Depois criei a variável predictions para salvar as previsões e então uso o conjunto de teste para testar o neurônio. Então calculo o erro no conjunto de teste.

```
# Normalização dos dados para que tenham média zero e variância unitária
X_train = (X_train - np.mean(X_train, axis=0)) / np.std(X_train, axis=0)
X_test = (X_test - np.mean(X_test, axis=0)) / np.std(X_test, axis=0)

model = Neuron(input_size=X_train.shape[1])

epochs = 100
lr = 0.01

for epoch in range(epochs):
    error = 0
    for x, y in zip(X_train, y_train):
        a = model.forward(x)
        model.backward(x, y, a, lr)
        error += abs(y - a)
    if error == 0:
        break

predictions = []
for x in X_test:
    a = model.forward(x)
    predictions.append(a)

predictions = np.array(predictions)

error = np.mean(predictions != y_test)

print('Erro no conjunto de teste:', error)
```

Resultado:

```
Erro no conjunto de teste: 0.02666666666666667
```

b) Treine um neurônio para classificar a classe Iris-virgínica como 1 e as demais como zero. Pare o algorítimo de treinamento após 100 épocas. Calcule o erro no conjunto de teste.
Fiz tudo igual a letra anterior, com exceção que mudei para definir a iris-virgínica como 1 e as demais como 0.

```
# Define a Iris-virgínica = 1, demais = 0
y[y != 2] = -1
y[y == 2] = 1
y[y != 1] = 0
```

Resultado:

```
Erro no conjunto de teste: 0.013333333333333334
```

c) Inicie os pesos aleatoriamente, cada peso tem um valor randômico uniformemente distribuído entre 0 e 1. Repita o processo das letras (a) e (b) 30 vezes para três valores distintos de taxa de aprendizagem 0,1; 1,0 e 10,0. Calcule a média e o desvio padrão das taxas de erro.
Aqui a diferença é a função train_and_test_neuron que vai nos ajudar com os valores da taxa de aprendizagem.

```
def train_and_test_neuron(X_train, y_train, X_test, y_test, lr, epochs=100):
    model = Neuron(input_size=X_train.shape[1])

    for epoch in range(epochs):
        for x, y in zip(X_train, y_train):
            a = model.forward(x)
            model.backward(x, y, a, lr)

    predictions = []
    for x in X_test:
        a = model.forward(x)
        predictions.append(a)

    predictions = np.array(predictions)

    error = np.mean(predictions != y_test)

    return error
```

Aqui definimos a taxa de aprendizagem e treinamos o neurônio com cada uma delas.

```
lrs = [0.1, 1.0, 10.0]
n_repetitions = 30
n_epochs = 100

for lr in lrs:
    errors = []
    for i in range(n_repetitions):
        error = train_and_test_neuron(X_train, y_train, X_test, y_test, lr, epochs=n_epochs)
        errors.append(error)

    mean_error = np.mean(errors)
    std_error = np.std(errors)

    print('LETRA A')
    print('Taxa de aprendizagem:', lr)
    print('Erro médio:', mean_error)
    print('Desvio padrão do erro:', std_error)
    print('\n')
```

Resultado:

```
LETRA A
Taxa de aprendizagem: 0.1
Erro médio: 0.020888888888888887
Desvio padrão do erro: 0.012260545977007752


LETRA A
Taxa de aprendizagem: 1.0
Erro médio: 0.016444444444444442
Desvio padrão do erro: 0.015273635802168518


LETRA A
Taxa de aprendizagem: 10.0
Erro médio: 0.0031111111111111114
Desvio padrão do erro: 0.005639367795755343


LETRA B
Taxa de aprendizagem: 0.1
Erro médio: 0.030222222222222223
Desvio padrão do erro: 0.013741888741102494


LETRA B
Taxa de aprendizagem: 1.0
Erro médio: 0.03733333333333333
Desvio padrão do erro: 0.015549205052920801


LETRA B
Taxa de aprendizagem: 10.0
Erro médio: 0.030222222222222216
Desvio padrão do erro: 0.0102896328024802
```

2) Utilizando o dataset Wine https://archive.ics.uci.edu/ml/datasets/wine e alguma biblioteca de redes neurais (sugestão, https://github.com/JeffersonLPLima/ANN, ver também o link com exemplo para a base Iris). Utilize 50% dos dados para treino e o restante para teste.

a) Ajuste os parâmetros da rede (número de épocas de treino, número de neurônios por camada etc.) para que a acurácia seja maior que 74%. Talvez os dados precisem de pré-processamento para corrigir escalas muito discrepantes.
Primeiro eu carreguei a base com o pandas, depois separei a classe dos atributos. Depois usei o LabelBinarizer para converter as 3 classes possíveis da base para colunas binárias. Depois separei em teste e treinamento. Usei a biblioteca sugerida, e coloquei 2 camadas densas, a primeira com 26 neurônios e a camada de saída de 3 neurônios, e a função de ativação softmax.

```
cols = ['class', 'alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflava

data = pd.read_csv('wine.data', header=None, names=cols)

data_x = data[cols[1:]]
data_y = data['class']

lb = preprocessing.LabelBinarizer()
data_y_transformed = lb.fit_transform(data_y)

data_train_x, data_test_x, data_train_y, data_test_y = train_test_split(data_x, data_y_transformed, test_size=0.5)

model = Sequential()
model.add(Dense(26, input_dim=(13)))
model.add(Dense(3))
model.add(Activation("softmax"))
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
model.fit(data_train_x, data_train_y, epochs=100, batch_size=1)

loss, accuracy = model.evaluate(data_test_x, data_test_y)

print("Acurácia = {:.2f}".format(accuracy))
```

Acurácia: 87%

b) Depois de a rede estar ajustada, realize 30 repetições de holdout para calcular a média e desvio padrão da acurácia.
Nessa letra o código é praticamente o mesmo, só fiz o ajuste para realizar as 30 repetições de holdout e depois calcular o desvio padrão e a média.

```
cols = ['class', 'alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_

data = pd.read_csv('wine.data', header=None, names=cols)

data_x = data[cols[1:]]
data_y = data['class']

lb = preprocessing.LabelBinarizer()
data_y_transformed = lb.fit_transform(data_y)

accuracies = []

for i in range(30):
  data_train_x, data_test_x, data_train_y, data_test_y = train_test_split(data_x, data_y_transformed, test_size=0.5)

  model = Sequential()
  model.add(Dense(26, input_dim=(13)))
  model.add(Dense(3))
  model.add(Activation("softmax"))
  model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
  model.fit(data_train_x, data_train_y, epochs=100, batch_size=1)

  loss, accuracy = model.evaluate(data_test_x, data_test_y)
  accuracies = np.append(accuracies, accuracy)

dev = accuracies.std()
med = np.average(accuracies)

print(f'Desvio Padrão = {dev}')
print(f'Média = {med}')
```

Desvio Padrão = 0.05733641523281246
Média = 0.8921348412831624

3) Considere o conjunto de dados Spiral com apenas dois atributos descritos na figura abaixo. Foram treinadas várias redes MLP com função de ativação Sigmoide, sendo 70% dos dados utilizados para treinamento e o restante para teste. Os resultados dos experimentos estão descritos a seguir.

a) Explique as taxas de acurácia de cada caso, comparando com outros casos.

b) Qual configuração você escolheria? Por quê? Você proporia uma configuração distinta? Por quê?

c) Replique os resultados destes experimentos. Calcule a média para 30 repetições em cada caso.

d) Implemente uma variação do Caso 3 que rode por 5.000 épocas. Mostre as curvas de erro tanto para o conjunto de treino como para o conjunto de teste a cada 100 épocas de treinamento.

e) Avalie outras três configurações da rede, buscando uma maior acurácia, variando os 5 parâmetros utilizados abaixo (número de época, taxa de aprendizagem, número de neurônios na 1a, 2a e 3a camadas escondidas).