Aprendizagem de Máquina
Prof. Tiago Buarque A. de Carvalho

Exercícios sobre Redes Neurais

1. (30 pontos) Utilizado a base Iris , https://archive.ics.uci.edu/ml/datasets/Iris. Os 25 primeiros exemplos de cada classe são o conjunto de treino, os outros exemplos são o conjunto de teste. Faça sua própria implementação de Neurônio Artificial.

   (a) Treine um neurônio para classificar a classe Iris-setosa como 1 e as demais como zero. Pare o algoritmo de treinamento com erro zero no conjunto de treino. Calcule o erro no conjunto de teste.

   erro maximo: 0

   pesos iniciais: 0

   (b) Treine um neurônio para classi car 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.

   (c) Inicie os pesos aleatoriamente, cada peso tem um valor randômico uniformemente distri buído 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.

2. (20 pontos) 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.

```
#   PRÉ-PROCESSAMENTO
#  Codifique rótulos de destino com valor entre 0 e n_classes-1.
encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
# le = preprocessing.LabelEncoder()
# >>> le.fit([1, 2, 2, 6])
# LabelEncoder()
# >>> le.classes_
# array([1, 2, 6])
# >>> le.transform([1, 1, 2, 6])
# array([0, 0, 1, 2]...)
# >>> le.inverse_transform([0, 0, 1, 2])
# array([1, 1, 2, 6])
```

```
# um array numpy (ou) um vetor que possui números inteiros que representam diferentes categorias,
# pode ser convertido em um array numpy (ou) uma matriz que possui valores binários e possui colunas iguais ao número de categorias no dados.
# Esta função retorna uma matriz de valores binários ('1' ou '0').
# Possui número de linhas igual ao comprimento do vetor de entrada e número de colunas igual ao número de classes.
bin_y = np_utils.to_categorical(encoded_Y)
```

✓ 0.1s

```
print(y)
print("\n")

print(encoded_Y)
print("\n")

print(bin_y)
```

✓ 0.1s

```
Output exceeds the size limit. Open the full output data in a text editor
0      1
1      1
2      1
3      1
4      1
      ..
173    3
174    3
175    3
176    3
177    3
Name: 0, Length: 178, dtype: int64


[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]


[[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
...
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]]
```

```
# Um modelo sequencial é apropriado para uma pilha simples de camadas onde cada camada tem exatamente um tensor de entrada e um tensor de saída.

model = Sequential()
# modelo sequencial de forma incremental
# a especificação de sua forma de entrada por meio do argumento input_dim
# Dense(38) é uma camada totalmente conectada com 38 unidades ocultas.
model.add(Dense(38, input_dim = 13))
# Função de ativação sigmoide
# Saída ε [0, 1] Derivada em [0; 0,25]
model.add(Activation("sigmoid"))
model.add(Dense(3))
# Softmax é uma função matemática que converte um vetor de números em um vetor de probabilidades,
# onde as probabilidades de cada valor são proporcionais à escala relativa de cada valor no vetor.
model.add(Activation("softmax"))
```

(b) Depois de a rede estar ajustada, realize 30 repetições de holdout para calcular a média e desvio padrão da acurácia.

```python
accuracyList = []
for i in range(30):
    treinoX,testeX,treinoY,testeY = train_test_split(x,bin_y, train_size=0.5,random_state=i)
    model.compile(optimizer= 'adam',loss= "binary_crossentropy", metrics=["accuracy"])
    model.fit(treinoX, treinoY, epochs=200, verbose= 0)

    loss, accuracy = model.evaluate(testeX,testeY, verbose= 0)

    accuracyList.append(accuracy)
```

```
Lista de acertos: [0.9438202381134033, 1.0, 0.9775280952453613, 0.966292142868042, 0.966292142868042, 0.9438202381134033, 0.9438202381134033, 0.9775280952453613, 0.9550561904907227,
0.9775280952453613, 0.9775280952453613, 0.9438202381134033, 0.9775280952453613, 1.0, 0.9775280952453613, 0.9887640476226807, 0.9887640476226807, 0.966292142868042,
0.9775280952453613, 0.9213483333587646, 0.966292142868042, 0.9887640476226807, 0.966292142868042, 0.9550561904907227, 0.9887640476226807, 0.9550561904907227, 0.966292142868042,
0.9775280952453613, 0.9775280952453613, 0.9775280952453613]


Desvio Padrão acuracia: 0.01822523596221791


Media da acuracia: 0.9696629285812378
```

Observação:

O tensorflow do https://github.com/JeffersonLPLima/ANN está com alguns problemas de compilação e warnings

```
2023-02-12 15:27:00.285534: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot
open shared object file: No such file or directory
2023-02-12 15:27:00.285643: W tensorflow/compiler/xla/stream_executor/cuda/cuda_driver.cc:265] failed call to cuInit: UNKNOWN ERROR (303)
2023-02-12 15:27:00.285713: I tensorflow/compiler/xla/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not appear to be running on this host (laisy-300E5K-300E5Q):
/proc/driver/nvidia/version does not exist
2023-02-12 15:27:00.289692: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x7f224c70f040> triggered tf.function retracing. Tracing is expensive
and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please
refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x7f224c5bf160> triggered tf.function retracing. Tracing is expensive
and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please
refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

3. (50 pontos) Considere o conjunto de dados Spiral com apenas dois atributos descritos na figura abaixo. Foram treinadas várias redes MLP com função de ativação Sigmóide, 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.

No caso 1, com acurácia 86,6%, é menor do que o caso 2 devido ao número de neurônios na 2 camada, em comparação ao caso 3 ela é maior devido ao número de neurônios na 2ª e 3ª camadas, no caso 4 ela é igual, mesmo o caso 4 tento mais neurônios nas 2ª e 3ª camadas e tendo o dobro de número de épocas.

No caso 2, com acurácia de 88,8%, é maior do que o caso 1 e 3, devido aos neurônios nas 2ª e 3ª camadas, o mesmo em relação ao caso 4, e também no caso 4 o número de épocas é o dobro.

No caso 3, com acurácia de 50,0%, é menor do que todos os outros casos, em relação aos casos 1 e 2, provavelmente devido ao número de neurônios na 3ª camada e no caso 4, ela é menor devido ao número de épocas que é a metade.

No caso 4, com acurácia de 86,6%, é igual ao caso 1 mesmo tendo mais neurônios nas 2ª e 3ª camadas e o dobro de épocas, é menor do que o caso 2, provavelmente devido ao número de neurônios na camada 3 e ao número de épocas, e é maior do que o caso 3, devido ao número de épocas, que nesse caso é o dobro.

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

Escolheria o caso 2, devido à alta acurácia.

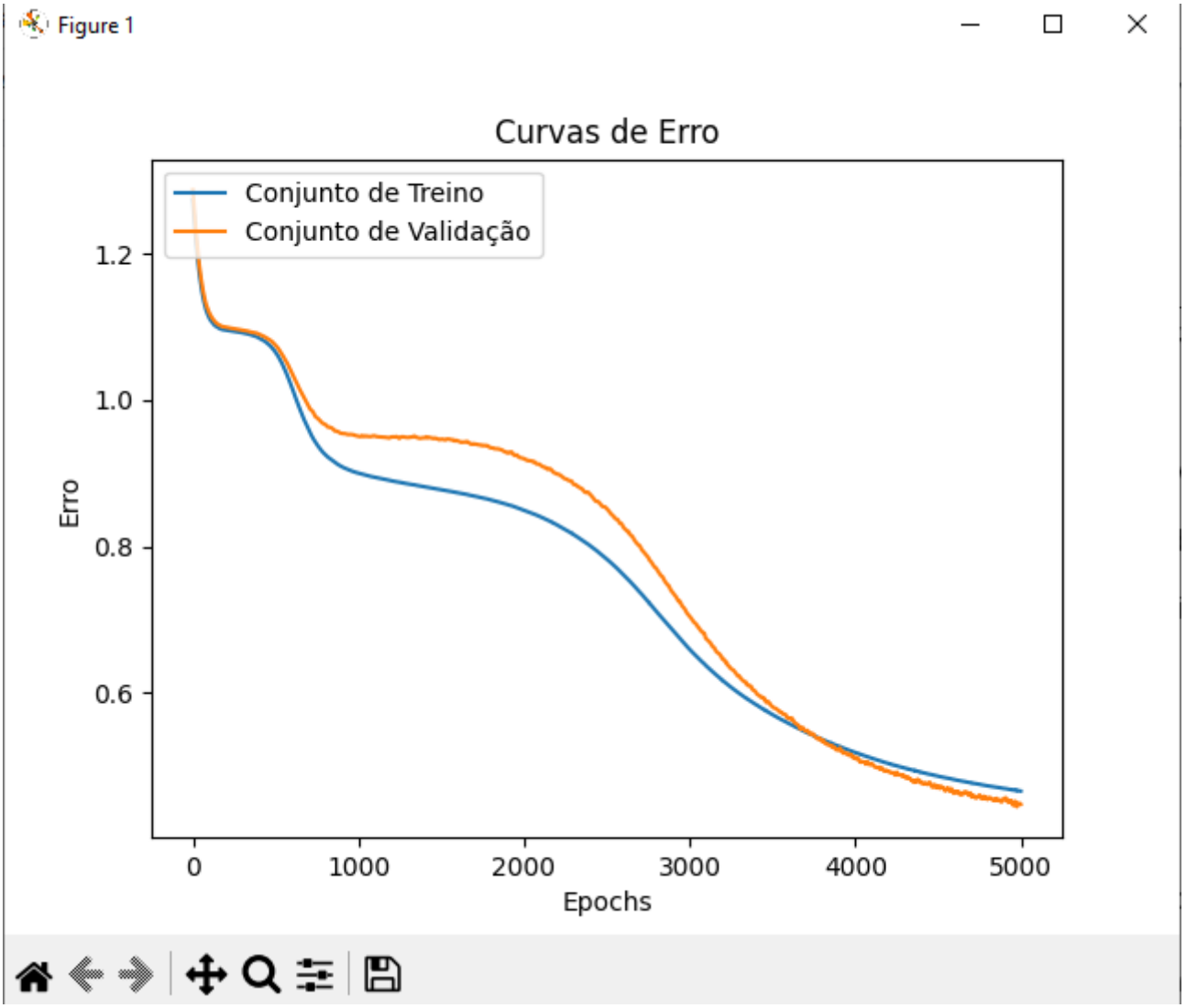
Proporia testar a configuração do caso 2, com uma alteração no numero de épocas, de 500 para 1000, aparentemente o número de épocas maior pode aumentar a acurácia.

(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.

```
X = dataset[:, 0:2].astype(float)
y = dataset[:, 2]

encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
bin_y = np_utils.to_categorical(encoded_Y)

X_train, X_test, y_train, y_test = train_test_split(X, bin_y, train_size=0.7, random_state=1)

model = Sequential()

model.add(Dense(4, input_shape=(None,2)))
model.add(Activation("sigmoid"))

model.add(Dense(4))
model.add(Activation("sigmoid"))

model.add(Dense(4))
model.add(Activation("sigmoid"))

model.add(Dense(3))
model.add(Activation("softmax"))

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
teste = model.fit(X_train, y_train, epochs=5000, batch_size=100, validation_split=0.3)

loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
```

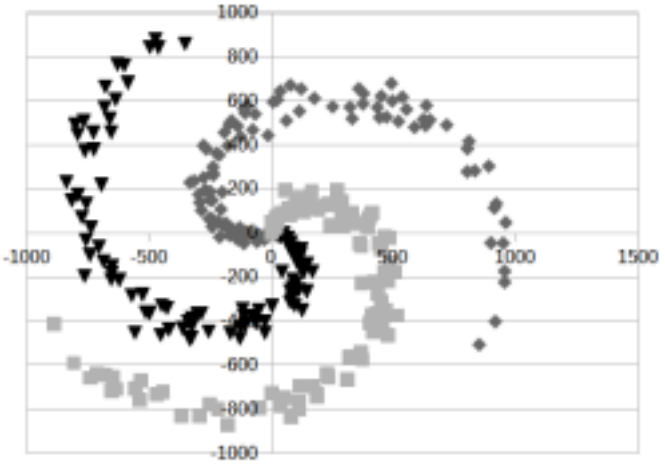
```
plt.plot(teste.history['loss'])
plt.plot(teste.history['val_loss'])
plt.title("Curvas de Erro")
plt.ylabel("Erro")
plt.xlabel("Epochs")
plt.legend(['Conjunto de Treino','Conjunto de Validação'],loc='upper left')
plt.show()
```



(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).

```
dataframe = pandas.read_csv(r"C:\Users\David\Downloads\Reconhecimento de Padrões\Reconhecimento de Padrões\Semana 10\Q3\spiral.csv", header=0)
dataset = dataframe.values

X = dataset[:, 0:2].astype(float)
y = dataset[:, 2]

encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
bin_y = np_utils.to_categorical(encoded_Y)

X_train, X_test, y_train, y_test = train_test_split(X, bin_y, train_size=0.5, random_state=1)

for i in [100, 1000, 10000]:
    clf2 = MLPClassifier(max_iter=i, learning_rate_init=0.4,verbose=0,hidden_layer_sizes=(4,3,5),activation='logistic')
    clf2= clf2.fit(X_train,y_train)
    pred = clf2.predict(X_test)
    taxa = accuracy_score(y_test,pred)*100
    print("Epochs: "+str(i)+" Learning: "+str(0.4));
    print("Taxa: ",taxa)
```

```
Epochs: 100 Learning: 0.4
Taxa:  0.0
Epochs: 1000 Learning: 0.4
Taxa:  85.33333333333334
Epochs: 10000 Learning: 0.4
Taxa:  77.33333333333333
```



	Caso 1	Caso 2	Caso 3	Caso 4
Número de épocas	500	500	500	1000
Taxa de Aprendizagem	0,3	0,3	0,3	0,3
Neurônios na 1ª camada escondida	4	4	4	4
Neurônios na 2ª camada escondida	0	4	4	4
Neurônios na 3ª camada escondida	0	0	4	4
Acurácia	86,6%	88,8%	50,0%	86,6%