Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Ciência da Computação
Inteligência Artificial Conexionista

Augusto André Souza Berndt 202000735

Exercício 7 - Problema da tireoide com RBF e probabilística.

Enunciado: Usando como base os arquivos e programas vistos em aula, implemente um código de Rede Neural RBF e Rede Neural Probabilística para o dataset da tireóide visto no exercício de classificação. Escreva um relatório relatando o trabalho feito e os códigos que você implementou.

1. Modelo *Radial basis function* (RBF): A biblioteca oficial Keras não disponibiliza uma camada RBF. Logo, para implementação dessa rede neural foi utilizado uma biblioteca pública disponível no github (https://github.com/PetraVidnerova/rbf_for_tf2) que faz a integração com a biblioteca Keras. Ou seja, o código anterior do exercício da tireóide foi facilmente adaptado para aplicação de uma camada RBF, como mostra a Figura 1.

```
from rbf_tf.rbflayer import RBFLayer, InitCentersRandom
from rbf_tf.kmeans_initializer import InitCentersKMeans
```

```
model = Sequential(
    [
        keras.Input(shape=(X_train.shape[-1],)),
        RBFLayer(700,
                 initializer=InitCentersKMeans(X_train),
                  betas=8.0),
        Dense(3, activation="softmax"),
    ]
)
model.summary()
```

Figura 1.

1.1 Função de ativação com "bug": Houveram diversas execuções que davam valores estranhos de treinamento, como mostra a Figura 2. Depois de muito tentar rodar e modificar o código, se reparou que o código que veio do exercício anterior estava com um otimizador de treinamento não ótimo que foi testado no exercício anterior (Ftrl), como também mostra a Figura 2.

```
model.compile(
#     optimizer=Adam(1e-3), loss="binary_crossentropy", metrics=metrics
      optimizer=Ftrl(learning_rate=0.001, name="Ftrl"), loss="binary_crossentropy", metrics=metrics
```

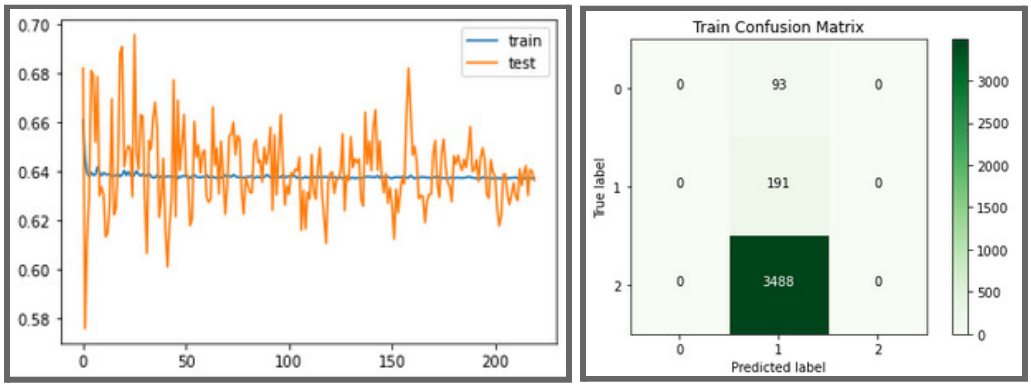


Figura 2.

1.2 Melhor, mas ainda limitado: Após consertar o erro, obtemos um treinamento positivo, como mostra a Figura 3, com 150 épocas apenas (estava se usando poucas épocas para testes mais rápidos).

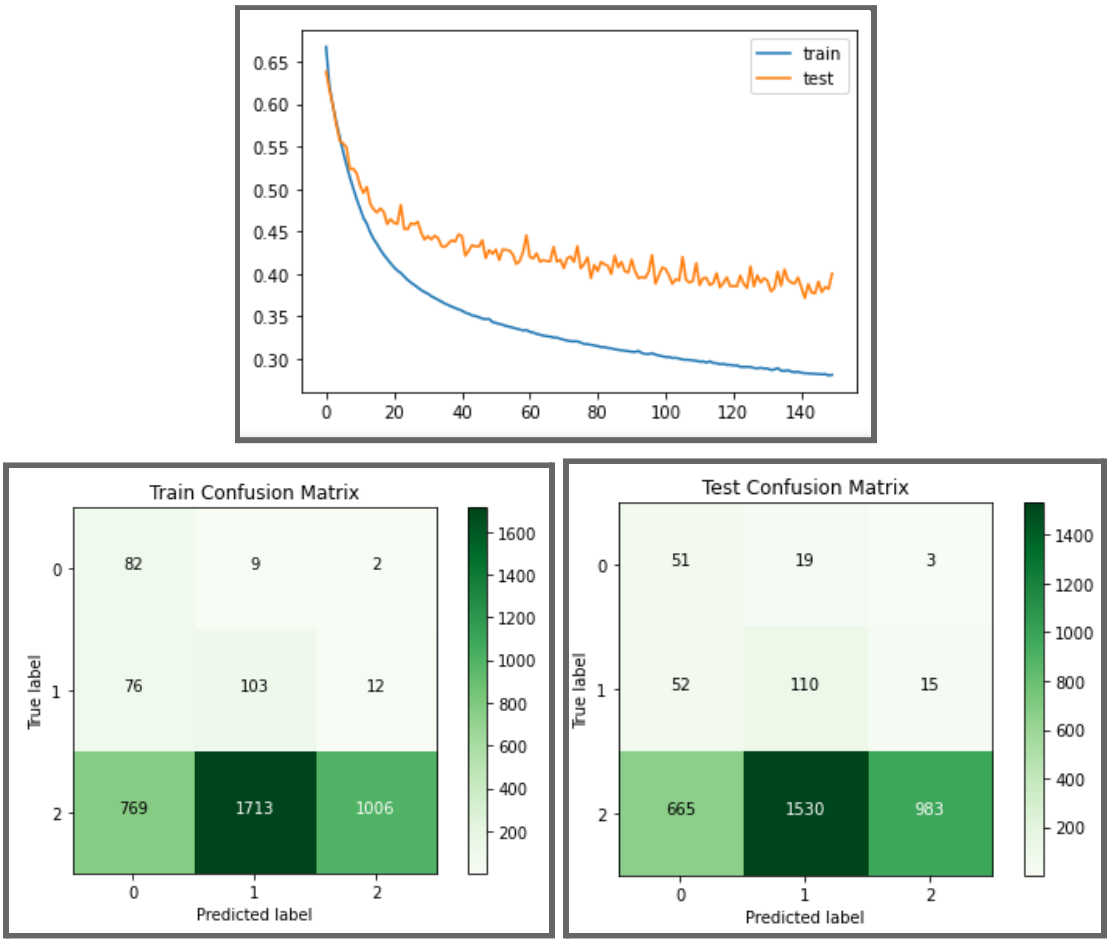


Figura 3.

Tenta-se então com mais épocas e a configuração mostrada na Figura 1. Com 700 neurônios, inicialização de K means, betas de 8.0 (tamanho dos radiais) e camada de saída com funções softmax. Os resultados de treinamento mostrados na Figura 4 mostram que definitivamente os hiperparâmetros precisam de um fine tuning.

	precision	recall	f1-score	support
Class 1	0.10	0.60	0.17	73
Class 2	0.08	0.59	0.15	177
Class 3	0.98	0.53	0.69	3178
accuracy			0.54	3428
nacro avg	0.39	0.58	0.34	3428
ghted avg	0.91	0.54	0.65	3428

Figura 4.

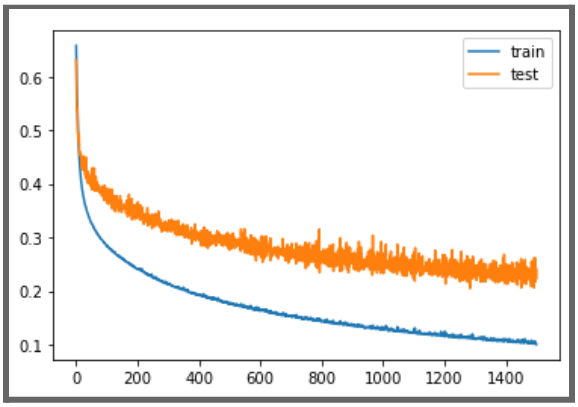
A configuração do modelo na Figura 5 foi reajustado diversas e várias vezes, porém sempre dando um treinamento e resultados de classificação bem limitados. Se tentou modificar empiricamente algumas da combinações a seguir sobre os hiperparâmetros para o treinamento da rede (estou citando os que lembro de ter utilizado, foram realmente muitas tentativas):

1. Número de neurônios - 10, 100, 500, 700, 800 e 900.
2. Parâmetro betas da função RBF - 2.0, 3.0, 5.0, 10.0, 12.0, 13.0 e 15.0
3. Inicializador do RBF - random ou aleatório
4. Função de ativação da camada de saída - softmax e sigmoid
5. Remoção ou não de alguns atributos (isto foi feito no exercício anterior da tireoide)
6. Número de épocas de treinamento - 100, 500, 1000 e 1500
7. Tamanho batch do treinamento - 100 e 200
8. Função de otimização do modelo - Adam (10^{-3}, 10^{-5}, 10^{-2}, 10^{-4}), Adamax (10^{-3},10^{-2}), Adagrad (10^{-3})

Mesmo com todas estas tentativas de diversos valores para os parâmetros de entrada, só se conseguiu obter um treinamento satisfatório modificando o modelo de erro de *binary_crossentropy* para **MSE!!!!!**

Notou-se que o erro inicial acaba dando valores mais baixos já inicialmente.

```
model = Sequential(
    [
        keras.Input(shape=(X_train.shape[-1],)),
        RBFLayer(900,
                 initializer=InitCentersKMeans(X_train),
                  betas=11.0),
        Dense(3, activation="softmax"),
    ]
)
model.summary()
```



Classification Report

	precision	recall	f1-score	support
Class 1	0.15	0.60	0.24	73
Class 2	0.11	0.51	0.18	177
Class 3	0.97	0.71	0.82	3178
accuracy			0.70	3428
macro avg	0.41	0.61	0.42	3428
weighted avg	0.91	0.70	0.78	3428

Figura 5.

1.3 Solução aceitável: As Figuras 6 e 7 mostram os valores já para um bom treinamento, depois de muitas e muitas tentativas. Infelizmente tentei alterar várias vezes 8 diferentes parâmetros até me dar de conta que a função custo utilizada que estava causando um treinamento ruim. As Figuras 3,4 e 5 são apenas alguns exemplos de vários que foram omitidos no relatório, onde o treinamento não foi satisfatório o bastante, oscilando entre 50% e 70%.

As Figuras 6 e 7 já demonstra um aprendizado satisfatório, com uma acurácia de 79%. O modelo RBF parece conseguir classificar corretamente o problema. Alguns atributos desnecessários foram removidos, assim como no exercício anterior da tireoide.

```
model = Sequential(
    [
        keras.Input(shape=(X_train.shape[-1],)),
        RBFLayer(900,
                 initializer=InitCentersRandom(X_train),
#                      initializer=InitCentersKMeans(X_train),
                 betas=12.0),
        Dense(3, activation="softmax"),
    ]
)
model.summary()
```

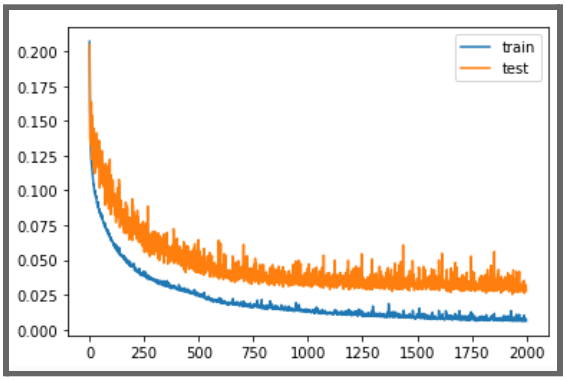
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rbf_layer_5 (RBFLayer)       (None, 900)               17100
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 2703
=================================================================
Total params: 19,803
Trainable params: 19,803
Non-trainable params: 0
```

```
model.compile(
#     optimizer=Adam(1e-3), loss="binary_crossentropy"
    optimizer=Adam(1e-3), loss="mse", metrics=metrics
#          optimizer=Ftrl(learning_rate=0.001, name="F
)

callbacks = [keras.callbacks.ModelCheckpoint("thyroid_

history = model.fit(
    X_train,
    y_train,
    batch_size=100,
    epochs=2000,
    verbose=1,
    callbacks=callbacks,
    validation_data=(X_validation, y_validation),
    class_weight=d_class_weights,
)
```

Figura 6 - configuração de hiperparâmetros.

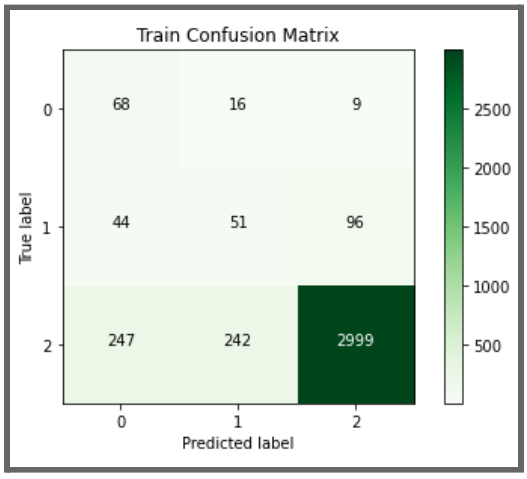



```
Classification Report

              precision    recall  f1-score   support

     Class 1       0.13      0.53      0.20        73
     Class 2       0.12      0.24      0.16       177
     Class 3       0.96      0.83      0.89      3178

    accuracy                           0.79      3428
   macro avg       0.40      0.54      0.42      3428
weighted avg       0.90      0.79      0.84      3428
```



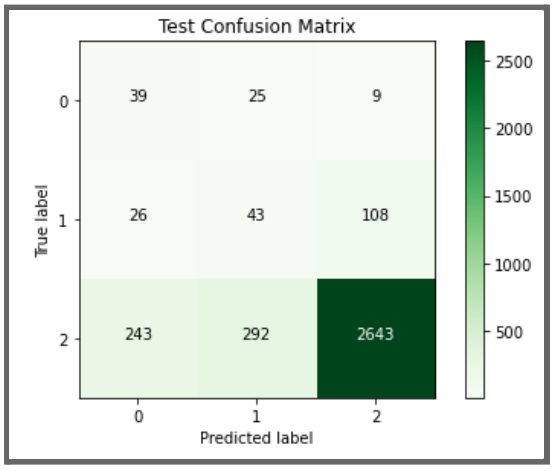


Figura 7 - resultados do treinamento.