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

Augusto André Souza Berndt 202000735

Exercício 5

Enunciado: No MATLAB use o "thyroid dataset " para desenvolver de maneira prática os conhecimentos vistos em aula sobre redes MLP. Faça experimentos com diferentes arquiteturas, taxas de aprendizado, algoritmos de treinamento, divisão dos dados, critérios de parada de treinamento, etc.... Escreva um relatório com os experimentos realizados e os resultados obtidos.
O Thyroid dataset também pode ser encontrado na UCI (https://archive.ics.uci.edu/ml/datasets/Thyroid+Disease) ou (http://networkrepository.com/thyroid-disease-thyroid0387.php)

1. Histogramas

Antes dos experimentos serem realizados com o código fornecido, se adicionou um código para viabilizar uma visualização do dataset além da tabela com o header do dataset. A Figura 1 mostra os histogramas do dataset dos pacientes com problema na tireoide, a última figura mostra a classificação (label) de cada indivíduo no dataset e cada uma de todas outras figuras mostra os dados para treinamento. Os dados de treinamento apresentam algumas *features* (colunas do dataset) tendenciosas, com grande maior parte dos valores iguais, por exemplo a feature 14 onde basicamente todos indivíduos possuem valor 0. A Figura 2 mostra os histogramas para o dataset de treinamento.

```
%matplotlib inline
import matplotlib.pyplot as plt
dataTrain.hist(bins=50, figsize=(20,15))
plt.show()
```

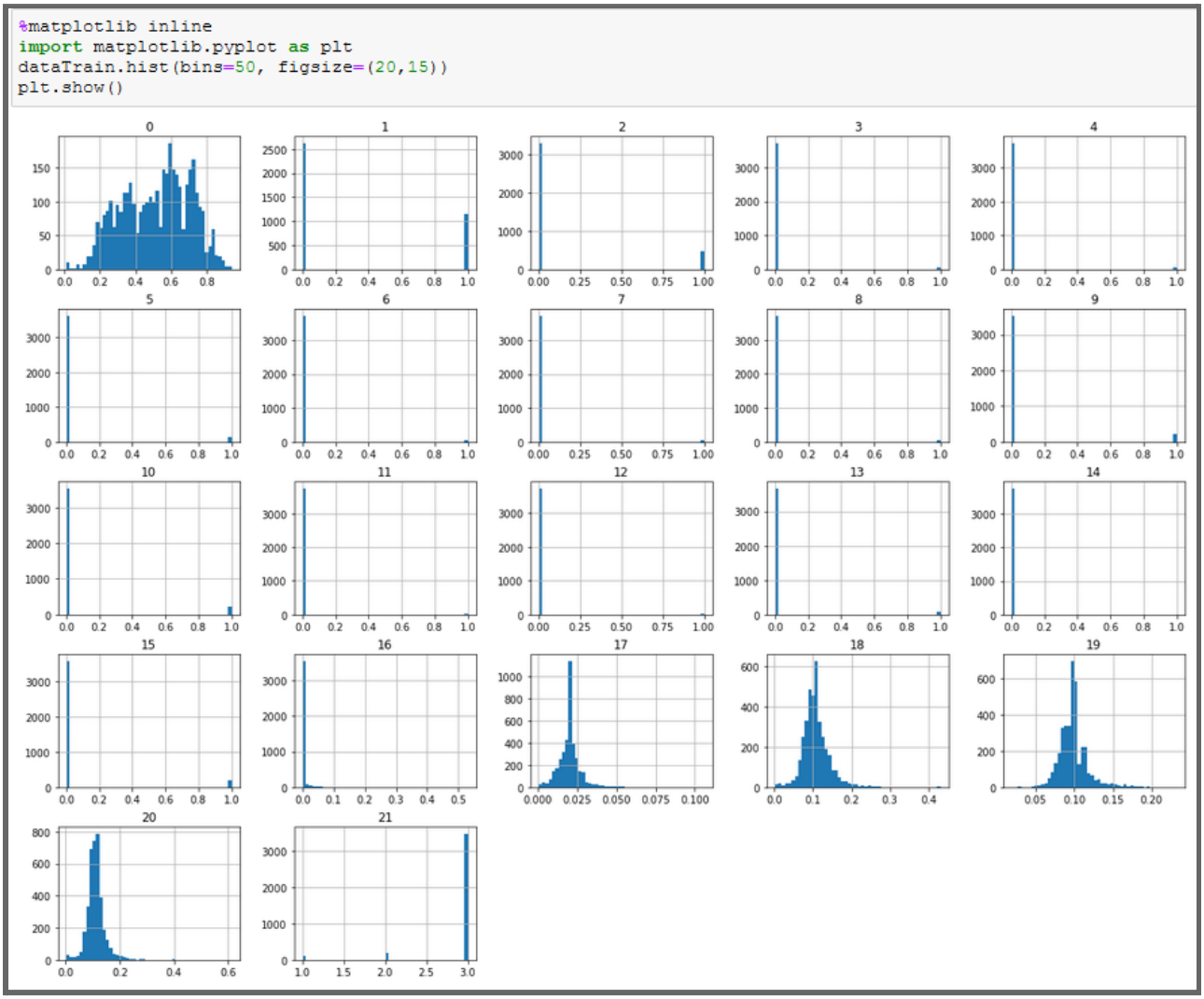


Figura 1 (histograma do dataset de **treinamento**).

```
%matplotlib inline
import matplotlib.pyplot as plt
dataTest.hist(bins=50, figsize=(20,15))
plt.show()
```

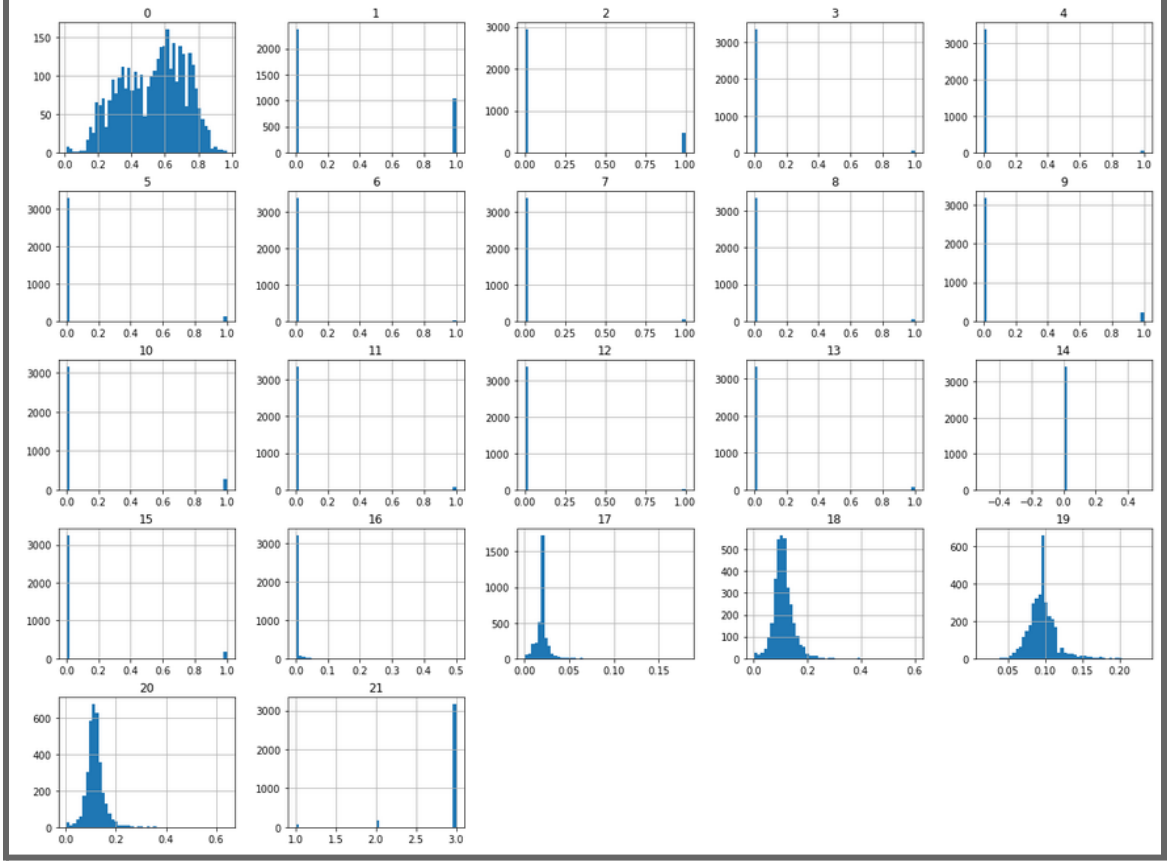


Figura 2 (histograma do dataset de **teste**).

2. Saídas originais

As saídas para valores originais como vieram no código são armazenadas aqui para comparações posteriores.

```
model = Sequential(
    [
        Dense(256, activation="relu", input_shape=(X_train.shape[-1],)
        ),
        Dense(256, activation="relu"),
        Dropout(0.3),
        Dense(256, activation="relu"),
        Dropout(0.3),
        Dense(3, activation="sigmoid"),
    ]
)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               5632
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 771
=================================================================
Total params: 137,987
Trainable params: 137,987
Non-trainable params: 0
```

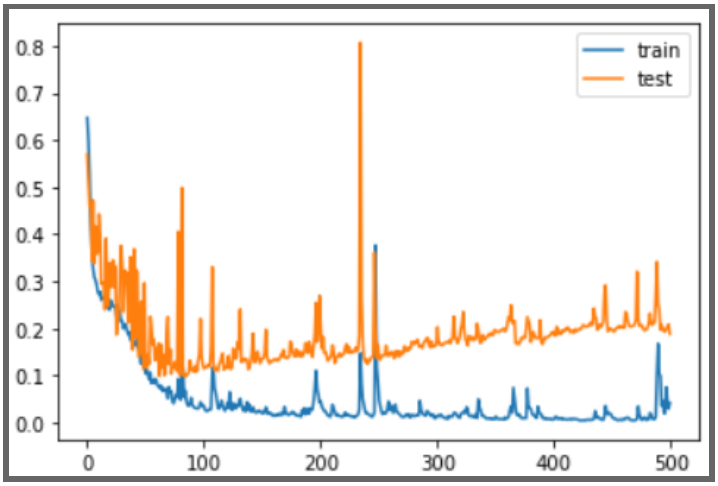
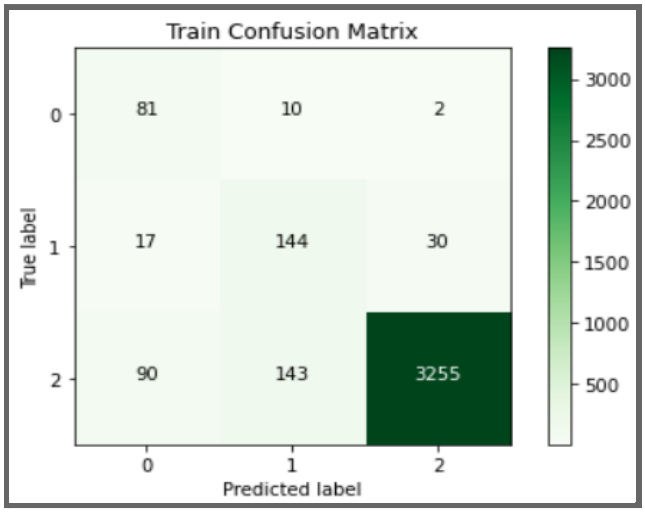
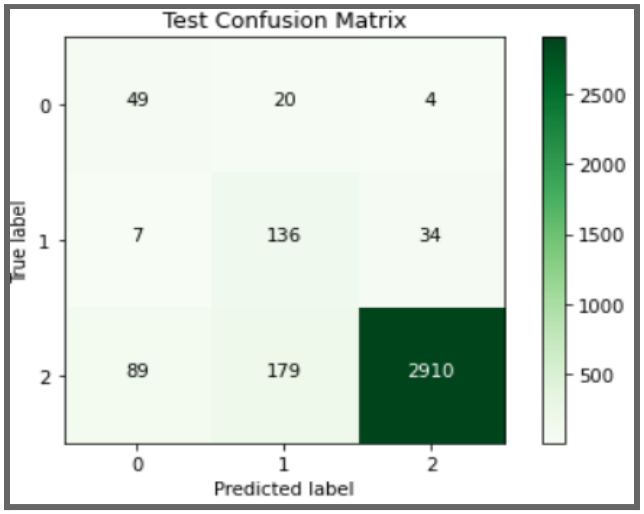


Figura 3 (modelo e treinamento original).



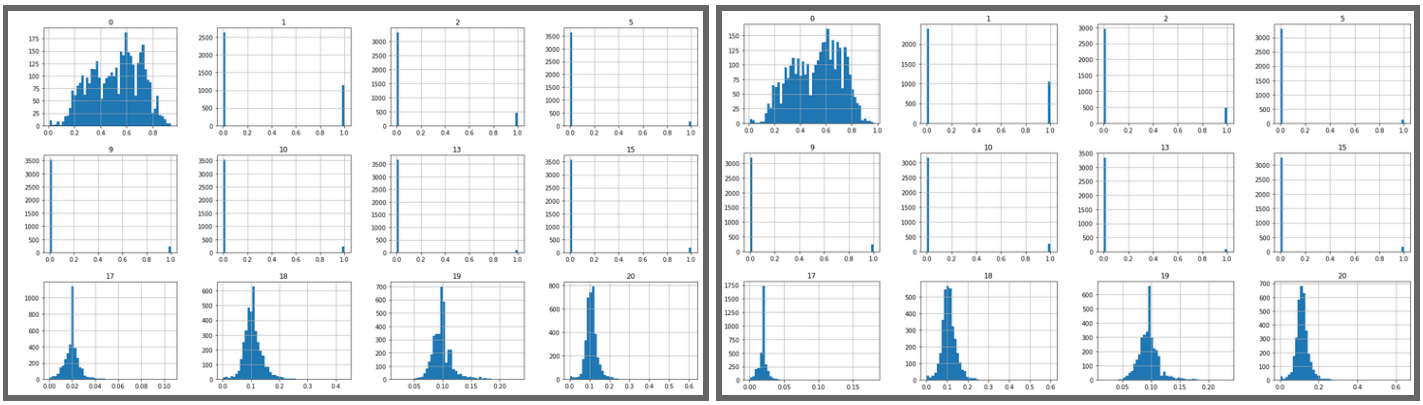



```
Classification Report

              precision    recall  f1-score   support

     Class 1       0.56      0.67      0.61        73
     Class 2       0.71      0.50      0.59       177
     Class 3       0.97      0.98      0.98      3178

    accuracy                           0.95      3428
   macro avg       0.75      0.72      0.73      3428
weighted avg       0.95      0.95      0.95      3428
```

Figura 4 (matrizes de confusão e métricas para conjunto de teste).

3. Remoção de features desnecessárias.

Se tentou remover algumas features que parecem desnecessárias, como foi sugerido em aula. Como podemos ver nas Figuras 2 e 3 existem features que sempre possuem o mesmo valor. Com isso as features removidas são apresentadas na Figura 5.

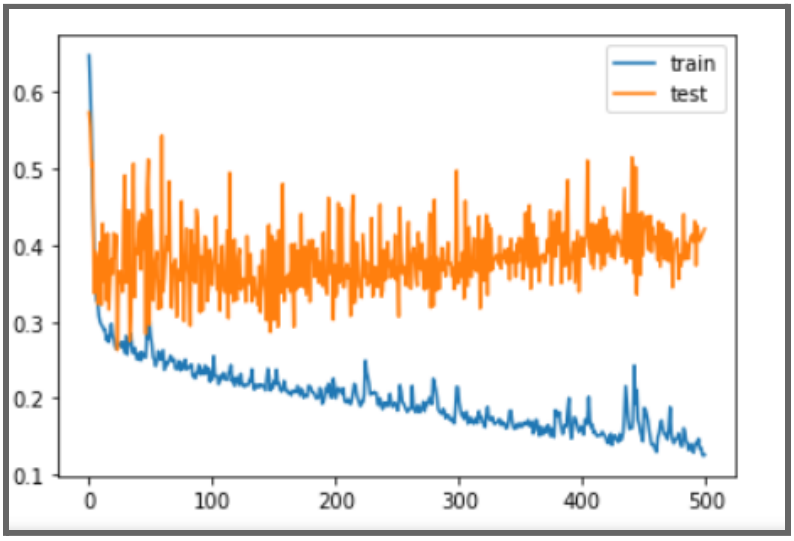


```
del dataTrain[22]
del dataTrain[23]

del dataTrain[3]
del dataTrain[4]
del dataTrain[6]
del dataTrain[7]
del dataTrain[8]
del dataTrain[11]
del dataTrain[12]
del dataTrain[14]
del dataTrain[16]
```

```
del dataTest[22]
del dataTest[23]

del dataTest[3]
del dataTest[4]
del dataTest[6]
del dataTest[7]
del dataTest[8]
del dataTest[11]
del dataTest[12]
del dataTest[14]
del dataTest[16]
```

Figura 5 (features removidas, treino e teste).

A Figura 6 demonstra o resultado obtido no treinamento com essa nova configuração do dataset de treino e teste. Visivelmente o número de gerações de 500 é muito para o aprendizado, uma vez que o treino e o teste começam a se afastar muito próximo da época 200 em diante. A matriz de confusão também indica que não houve uma melhoria no aprendizado, uma vez que o modelo confundido vários exemplares entre as classes 1 e 2. O desempenho na Figura 4 foi melhor. Foi tentado treinar o modelo com 200 epocas na tentativa de reduzir o ovefitting, porém o aprendizado segue ruim, como mostra a Figura 7.



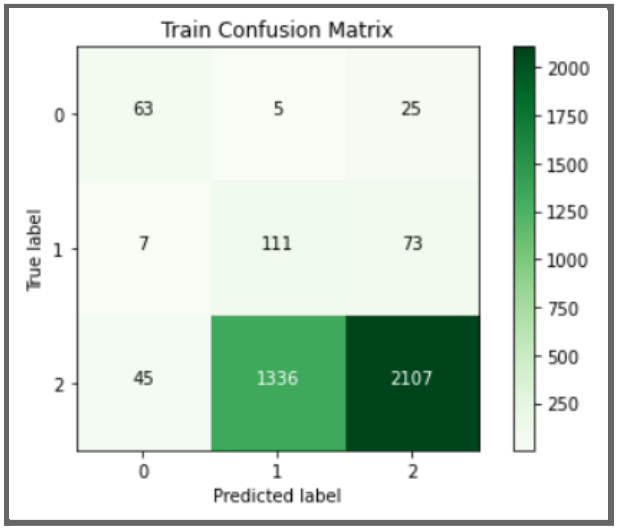


Figura 6 (treinamento e matriz de confusão para treino **500 épocas).**

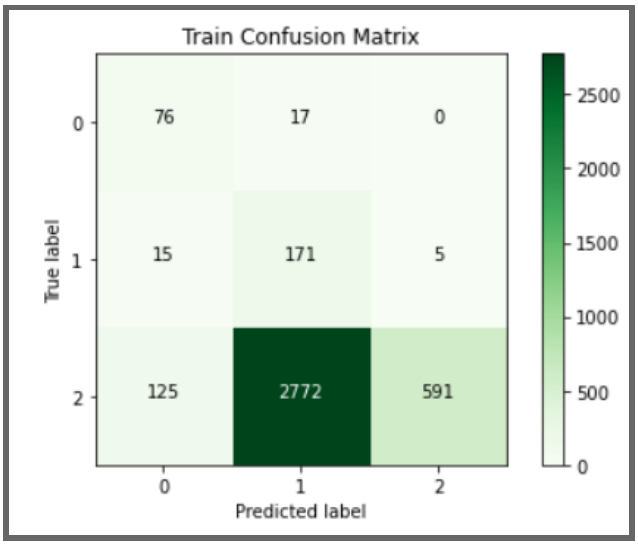
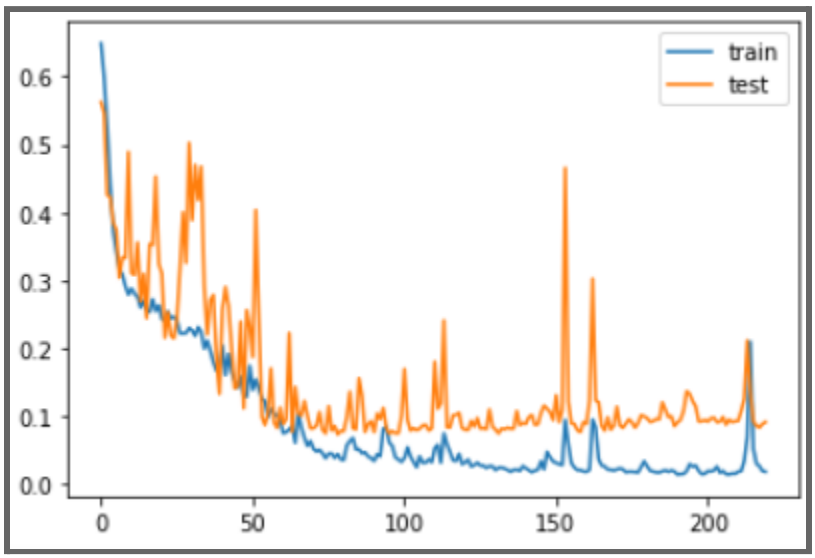
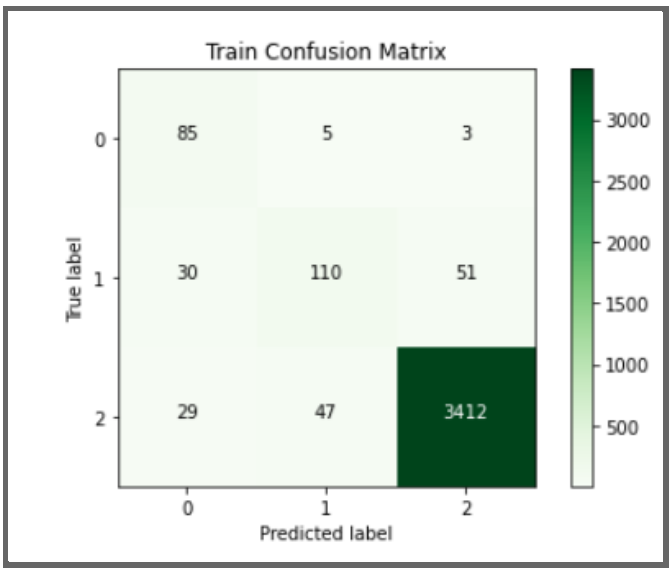


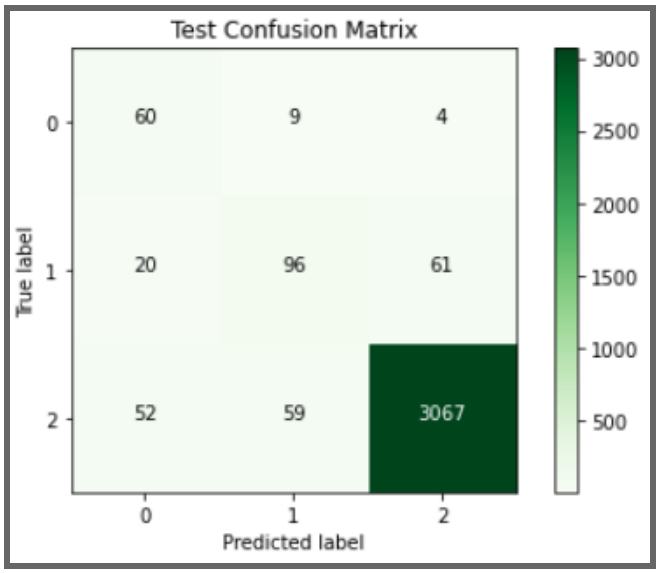
Figura 7 (matriz de confusão para treino **200 épocas).**

Tentaremos então uma seleção de features diferente, no qual se mostrou melhor do que a original. Aparentemente algumas features parecem desnecessárias, como mostra a Figura 2. Porém se deve ter cuidado, pois algumas features importantes foram removidas anteriormente, já que o aprendizado claramente piorou, como mostra as Figuras 5, 6 e 7.

A Figura 8 demonstra o treinamento para uma nova configuração do dataset, com 200 épocas e a remoção das features 11, 12 e 14, além da 22 e 23 que já haviam sido removidas. O treinamento se mostrou bem satisfatório, comparando as matrizes de confusão de teste entre a Figura 8 com a nova configuração e a Figura 4, tivemos um aprendizado consideravelmente melhor para os labels 0 e 2, porém o label 1 o modelo se saiu um pouco pior. Por mais que a matriz de confusão do novo dataset se mostrou melhor, as métricas de aprendizado não se mostraram vantajosas, com o F1 score, por exemplo, as métricas apresentadas são para o conjunto de teste.







Classification Report

	precision	recall	f1-score	support
Class 1	0.45	0.82	0.59	73
Class 2	0.59	0.54	0.56	177
Class 3	0.98	0.97	0.97	3178
accuracy			0.94	3428
macro avg	0.67	0.78	0.71	3428
weighted avg	0.95	0.94	0.94	3428

Figura 8 (treinamento com dataset reduzido).

Se fez então o treinamento com o dataset reduzido, mas agora para 500 épocas, agora sim os resultados são melhores do que os originais, mostrando uma boa escolha de feature selection, como mostram os dados da Figura 9, todos melhores ou iguais aos apresentados na Figura 8.

Classification Report

	precision	recall	f1-score	support
Class 1	0.48	0.84	0.61	73
Class 2	0.72	0.59	0.65	177
Class 3	0.98	0.97	0.98	3178
accuracy			0.95	3428
macro avg	0.73	0.80	0.75	3428
weighted avg	0.96	0.95	0.95	3428

Figura 9 (métricas de treino para conjunto reduzido de teste).

3. Arquiteturas

Agora tentaremos melhorar o modelo a propondo uma arquitetura diferente para o modelo, ou seja, modificando as camadas da rede. A alteração de feature selection será mantida, assim como as 500 épocas de treinamento.

A Figura 3 demonstra os tipos de camadas utilizados, são 3 camadas completamente conectadas com funções de ativação *relu* e uma camada de saída com função de ativação *sigmoid.* Entre a segunda e terceira camadas ReLu são aplicadas duas camadas *dropout.*

Como consta na documentação do Keras sobre a camada *dropout* (novidade pra mim): "*The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged*." Ou seja, ela exclui alguns sinais de ativação durante o treinamento, de maneira aleatória, na intenção de diminuir o *overfitting,* ao que parece procurando por sinais desnecessários.

Para propor uma arquitetura do modelo diferente iremos manter as camadas de dropout, mas modificamos as demais camadas. A documentação do Keras apresenta as seguintes opções de função de ativação:

- Relu
- Sigmoid
- Softmax
- **Sofplus**
- **Softsign**
- Tanh - Hyperbolic tangent
- **Selu - Scaled Exponential Linear Unit**
- **Elu - Exponential Linear Unit**
- Exponential

Marcado em negrito as que eu ainda não conhecia. O keras ainda dá a opção de criar uma ativação *custom* através de um tensor. A Figura 10 mostra algumas destas funções.

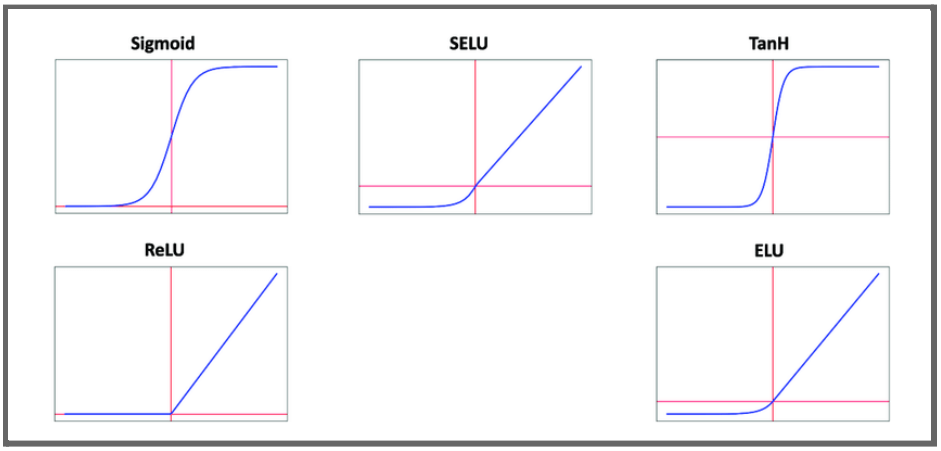
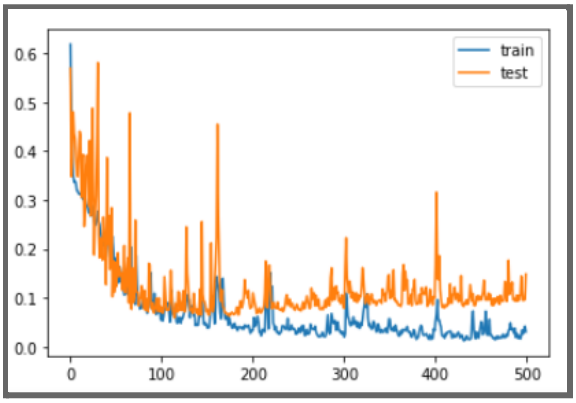


Figura 10 (algumas funções de ativação disponíveis no keras).

A Figura 11 mostra os resultados para um treinamento substituído as 3 relus para SELU e a camada de saída mantida como sigmoid, como podemos ver, as métricas de treinamento com as camadas com esta função obtiveram consideravelmente melhores resultados em todas as métricas, principalmente para macro average, acurácia e os F1 scores das classes, esta função de ativação basicamente deu uma boa concertada nos F1 scores que estavam baixíssimos, foram de 0.61, 0.65 e 0.98 para 0.77, 0.74 e 0.98. A Figura 11 também mostra como o aprendizado com 500 epocas foi o bastante, uma vez que o custo entre teste e treino recém começou a se afastar já nas 500 épocas.

```
model = Sequential(
    [
        Dense(256, activation="selu", input_shape=(X_train.shape[-1],)
        ),
        Dense(256, activation="selu"),
        Dropout(0.3),
        Dense(256, activation="selu"),
        Dropout(0.3),
        Dense(3, activation="sigmoid"),
    ]
)
model.summary()
```



Classification Report

	precision	recall	f1-score	support
Class 1	0.66	0.93	0.77	73
Class 2	0.74	0.74	0.74	177
Class 3	0.99	0.98	0.98	3178
accuracy			0.97	3428
macro avg	0.80	0.88	0.83	3428
weighted avg	0.97	0.97	0.97	3428

Figura 11 (resultados de treinamento com funções SELU, reporte para teste).

5. Algoritmo de Treinamento

Tentou-se modificar como a função custo é calculada, mas não os resultados não foram melhores do que com binary crossentropy. Como mostra a Figura 12.

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

Classification Report

	precision	recall	f1-score	support
Class 1	0.35	0.92	0.51	73
Class 2	0.71	0.67	0.69	177
Class 3	0.99	0.96	0.97	3178
accuracy			0.94	3428
macro avg	0.68	0.85	0.72	3428
weighted avg	0.96	0.94	0.95	3428

Figura 12 (treinamento com função custo **Mean squared error MSE**).

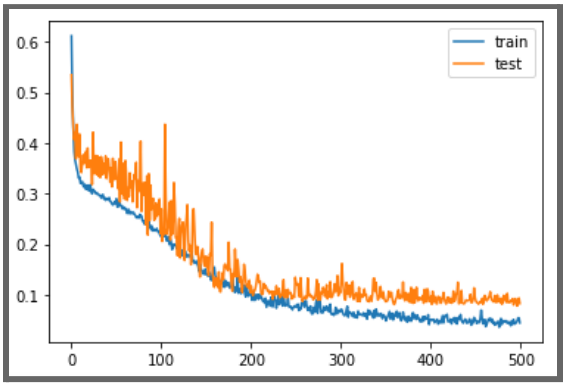
Existem também várias outras opções além do otimizador ADAM, como mostra a Figura 13, retirado da documentação do Keras. Como mostra a Figura 14, a otimização ADAMAX não demonstrou resultados melhores para o treinamento. As 500 epocas demoraram um pouco mais para terminarem com esta opção de otimização.

Available optimizers

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

Figura 13 (possíveis funções de otimização).

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

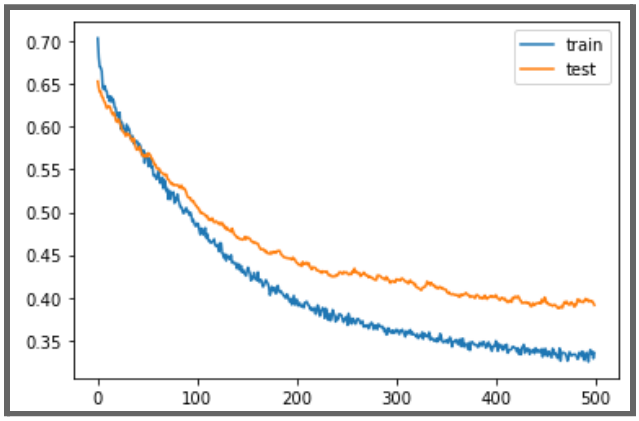


Classification Report

	precision	recall	f1-score	support
Class 1	0.47	0.93	0.63	73
Class 2	0.82	0.62	0.71	177
Class 3	0.98	0.97	0.98	3178
accuracy			0.96	3428
macro avg	0.76	0.84	0.77	3428
weighted avg	0.96	0.96	0.96	3428

Figura 14 (treinamento com otimização ADAMAX).

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



Classification Report

	precision	recall	f1-score	support
Class 1	0.13	0.90	0.22	73
Class 2	0.07	0.69	0.12	177
Class 3	0.99	0.34	0.51	3178
accuracy			0.37	3428
macro avg	0.39	0.65	0.28	3428
weighted avg	0.92	0.37	0.48	3428

Figura 15 (treinamento com otimização ADAGRAD).

A Figura 15 mostra como a opção ADAGRAD também não foi capaz de encontrar melhores resultados de treinamento do que a função ADA, a otimização ADAMAX ainda foi melhor que esta última. A opção Ftrl deu erro por algum motivo, como mostra a última Figura.

```
c:\users\user\appdata\local\programs\python\python38\lib\site-packages\skl
earn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\users\user\appdata\local\programs\python\python38\lib\site-packages\skl
earn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\users\user\appdata\local\programs\python\python38\lib\site-packages\skl
earn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision an
d F-score are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\users\user\appdata\local\programs\python\python38\lib\site-packages\skl
earn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision an
d F-score are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\users\user\appdata\local\programs\python\python38\lib\site-packages\skl
earn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision an
d F-score are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```