**I. Pen-and-paper**

1. Cálculos da propagação:

Cálculo dos deltas:

Atualização dos pesos e dos bias:

1. Cálculos da propagação:

Cálculo dos deltas:

Atualização dos pesos e dos bias:

**II. Programming and critical analysis**

1. Após testar o modelo pretendido para valores de 0.1, 1 e 10 no parâmetro de regularização (alpha), obtivemos os seguintes valores de eficácia:

|  |  |  |  |
| --- | --- | --- | --- |
|  | alpha=0.1 | alpha=1 | alpha=10 |
| Sem *early stopping* | 93.71% | 93.56% | 95.76% |
| Com *early stopping* | 89.30% | 89.16% | 91.06% |

Decidimos assim fixar um alpha de valor 10, uma vez que foi a opção que permitiu uma maior eficácia, na presença e ausência de *early stopping.*

Obtemos, de seguida, as matrizes de confusão para ambos os casos através da soma das matrizes de confusão de cada um dos 5 *folds* utilizados.

Treino sem *early stopping* Treino com *early stopping*

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Valores Reais | |
| P | N |
| Valores Previstos | P | 426 | 18 |
| N | 11 | 228 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Valores Reais | |
| P | N |
| Valores Previstos | P | 384 | 60 |
| N | 1 | 238 |

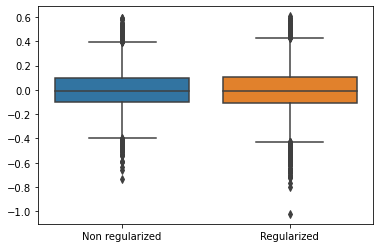
Podemos observar que ao aplicar a técnica de *early stopping,* a eficácia do modelo acaba por piorar sobretudo devido à menor quantidade de verdadeiros positivos e maior quantidade de falsos positivos, comparativamente à solução sem *early stopping*.

Uma das razões que pode levar à existência destas diferenças é o facto de, ao aplicar *early stopping*, a estabilização do erro no conjunto de validação se poder tratar de um mínimo local para o erro do modelo, o que leva à execução de um número de épocas inferior ao necessário para atingir o mínimo absoluto do mesmo erro.

Para além disso, estas diferenças podem ainda dever-se ao facto de, na aplicação de *early stopping*, parte do conjunto de dados de treino ser utilizado como conjunto de validação o que leva a uma menor dimensão do conjunto de dados de treino, o que consequentemente pode levar a uma diminuição da eficácia do modelo.

1. Após testar o modelo de regressão com regularização para valores de 0.1, 1 e 10 no parâmetro de regularização (alpha), verificamos que a soma dos erros quadrados do modelo no conjunto de treino era menor para o valor 0.1.

Logo, fixando esse parâmetro para o modelo com regularização, obtivemos a seguinte distribuição dos resíduos para um modelo com regularização e para um modelo sem regularização.



De forma a minimizar o erro dos modelos podemos adotar estratégias tais como: aumentar a dimensão do conjunto de treino permitindo uma maior abrangência de dados e um menor erro; selecionar as variáveis mais correlacionadas com a variável de output; alterar o número de camadas internas da rede bem como o número de percetrões de forma a descobrir a arquitetura que mais se adequa ao problema em questão; e, tal como feito neste exercício, testar diferentes níveis de regularização de maneira a encontrar um nível que permita um ajustamento razoável aos dados evitando ainda o *overfitting*.

**III. APPENDIX**

**import** numpy **as** np

**import** seaborn **as** sns

**from** sklearn.neural\_network **import** MLPClassifier, MLPRegressor

**from** sklearn **import** model\_selection

**from** sklearn.metrics **import** confusion\_matrix

**import** pandas **as** pd

**from** scipy.io **import** arff

data = arff.loadarff('breast.w.arff')

df = pd.DataFrame(data[0])

df.dropna(inplace=True)

df.replace(b'benign', 0, inplace=True)

df.replace(b'malignant', 1, inplace=True)

data = df.drop(["Class"],axis=1).values

target = df["Class"].values

fold = model\_selection.KFold(n\_splits=5, shuffle=True, random\_state=0)

conf\_matrix1 = np.matrix([[0,0],[0,0]])

conf\_matrix2 = np.matrix([[0,0],[0,0]])

**for** train\_filter, test\_filter **in** fold.split(data):

data\_train, data\_test, target\_train, target\_test = data[train\_filter], data[test\_filter], target[train\_filter], target[test\_filter]

mlp1 = MLPClassifier(hidden\_layer\_sizes=[3,2],alpha=10,shuffle=True,random\_state=0).fit(data\_train, target\_train)

mlp2 = MLPClassifier(hidden\_layer\_sizes=[3,2],alpha=10,shuffle=True,random\_state=0,early\_stopping=True).fit(data\_train, target\_train)

conf\_matrix1 = conf\_matrix1 + confusion\_matrix(target\_test, mlp1.predict(data\_test))

conf\_matrix2 = conf\_matrix2 + confusion\_matrix(target\_test, mlp2.predict(data\_test))

**print**(conf\_matrix1)

**print**(conf\_matrix2)

data = arff.loadarff('kin8nm.arff')

df = pd.DataFrame(data[0])

df.dropna(inplace=True)

data = df.drop(["y"],axis=1).values

target = df["y"].values

fold = model\_selection.KFold(n\_splits=5, shuffle=True, random\_state=0)

residuals1, residuals2 = [], []

**for** train\_filter, test\_filter **in** fold.split(data):

data\_train, data\_test, target\_train, target\_test = data[train\_filter], data[test\_filter], target[train\_filter], target[test\_filter]

mlp1 = MLPRegressor(hidden\_layer\_sizes=[3,2],alpha=0,shuffle=True,random\_state=0).fit(data\_train, target\_train)

mlp2 = MLPRegressor(hidden\_layer\_sizes=[3,2],alpha=0.1,shuffle=True,random\_state=0).fit(data\_train, target\_train)

residuals1 += **list**(mlp1.predict(data\_test) - target\_test)

residuals2 += **list**(mlp2.predict(data\_test) - target\_test)

df = pd.DataFrame(data={"Non regularized": residuals1, "Regularized": residuals2})

ax = sns.boxplot(data=df)

**END**