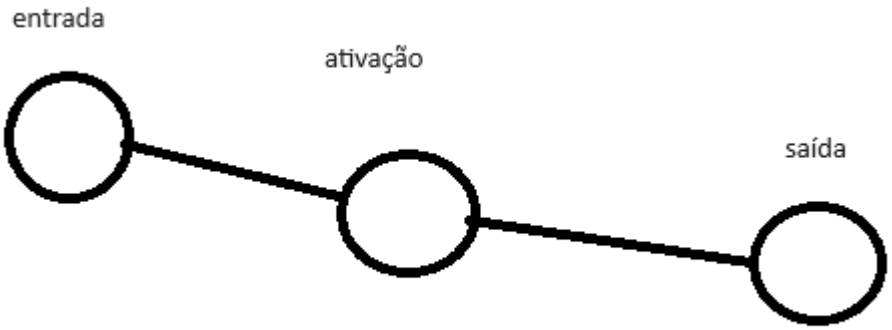
Luiz Felipe Nagatani Rabello

Relatório LAMIA - Módulo 12

Seção 3

Perceptron de uma camada

O curso se inicia explicando o conceito de neurônios artificiais, tal qual é representado por grafos ponderados, onde cada entrada possui um peso, todas estas entradas são somadas e processadas em uma função de ativação.



Redes Multicamada

Se diferenciam dos perceptrons de uma camada pois além de possuírem a camada de entrada e a de saída, possuem camadas ocultas entre elas, utilizadas em casos impossíveis de serem separados linearmente como por exemplo no caso de operadores XOR. No início do treinamento da rede, os pesos são inicializados com valores aleatórios pequenos e serão reajustados durante o treinamento.

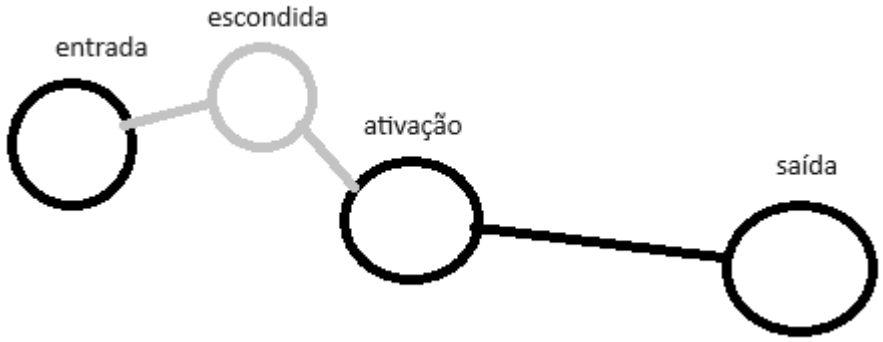
Para calcular o erro existem algumas ferramentas:

A mais simples é o algoritmo erro= respCorreta- respCalculada, e realizar isso para cada ponto. Isto é executado por diversas épocas até que o valor do erro seja suficientemente pequeno.

Para certificar que o erro irá para o mínimo global e não para o local, é utilizado o método do gradiente descendente, onde se encontrará uma medida onde a combinação de pesos resulta no menor erro possível

Para atualizar os pesos é calculado o delta, onde o valor de delta será acrescentado ou subtraído do peso na próxima época. Este processo, executado da saída até a entrada, é chamado de backpropagation, e utiliza os deltas e da taxa de aprendizagem.

O bias existe para garantir flexibilidade ao modelo no treinamento do mesmo e para que os pesos não estejam zerados quando as entradas forem 0, nele se insere valores constantes em cada camada, inclusive na de saída



Seção 4

Nesta seção, é utilizada a database breast cancer para criar um modelo que classifique tumores benignos e malignos utilizando classificação binária.

Ao prosseguir para a classificação da rede, ele mostra que é sequencial, pois segue da entrada para a saída, e densa, pois a cada nodo está ligado com todos os nodos da camada seguinte.

Aborda-se a criação e a estruturação do modelo, assim como o “tuning”, procedimento realizado para melhorar a precisão da máquina.

Para melhorar a precisão do modelo se viu necessário mudar a quantidade de épocas, dropout das camadas e a quantidade de CVs, resultando em um tempo de execução um pouco maior porém obtendo uma precisão de 88% em um teste realizado.

```
lef criaRede():
    # creating neurons
    clas = Sequential()
    clas.add(Dense(units=16, activation='relu',
                           kernel_initializer='random_uniform', input_dim = 30))
    clas.add(Dropout(0.4))
    clas.add(Dense(units=16, activation='relu',
                           kernel_initializer='random_uniform'))
    clas.add(Dropout(0.4))
    clas.add(Dense(units=1, activation='sigmoid'))
```

```
print(f'Média: {res.mean()}  Erro: {res.std()}')
```

```
Média: 0.8822681704260651  Erro: 0.03921369806735092
```

```
Dropout         Precisão
0.2               0.86          } testes realizados com 50 epocas e 10 cv
0.4               0.87

O melhor resultado foi com 0.4 de dropout em todas as camadas com 100 epocas, resultando 88.22% de precisão
```

Seção 5

Na seção 5 é utilizada classificação multiclasse para treinar um modelo com 3 tipos de flores e classificar a base de testes nesses 3 tipos.

Como exercício foi necessário realizar tuning no modelo, utilizando o método de tuning usando gridsearchCV(técnica para encontrar a melhor combinação de parâmetros de um modelo) com 5 CVs

```
[*]: params = {'batch_size' :[10, 30],
              'epochs':[1000, 3000], 'optmizer': ['adam'],
              'loss_function':['categorical_crossentropy'],
              'kernel_initializer':['random_uniform', 'normal'],
              'activation' : ['relu'],
              'neurons' : [4, 8]}
     grid_search = GridSearchCV(estimator = clas, param_grid = params, scoring = 'accuracy', cv = 5)
```

```
[15]: media = res.mean()
      desvio = res.std()
      print(f'média = {media}, desvio = {desvio}')
```

```
média = 0.9666666666666668, desvio = 0.061463629715285906
```

Seção 6

É utilizada regressão para prever valor de carros usados com base em suas peças e características, possuindo somente uma saída, se diferenciando da seção seguinte.

Para trabalhar com regressão e resultados categóricos, se viu adequado utilizar encoding, primeiro o label encoding para transformar colunas de “nomes” em colunas de números como por exemplo mudar todas as ocorrências de volkswagen de uma coluna chamada marca para 1.

Em seguida foi utilizado o one hot encoding, onde se transforma uma coluna em várias colunas com valores binários para acelerar o processamento

```
from sklearn.compose import ColumnTransformer

ct = ColumnTransformer([
    ("c0", OneHotEncoder(), [0]),
    ("c1", OneHotEncoder(), [1]),
    ("c3", OneHotEncoder(), [3]),
    ("c5", OneHotEncoder(), [5]),
    ("c8", OneHotEncoder(), [8]),
    ("c9", OneHotEncoder(), [9]),
    ("c10", OneHotEncoder(), [10]),

    ], remainder='passthrough')
```

Seção 7

Nesta seção é utilizada regressão multiclasse para prever quantidade de vendas de jogos em determinadas regiões, é utilizada a multiclasse pois apesar de utilizarem os mesmos parâmetros, para cada saída os parâmetros têm pesos diferentes. Permitindo que se busque valores para cada nodo de saída.

```
prediction_na, prediction_eu, prediction_jp = regressor.predict(predictors)
```

```
9/9 [==============================] - 0s 1ms/step
```

```
sales_na[0:10]
```

```
array([41.36, 15.68, 15.61, 11.28, 13.96, 14.44,  9.71,  8.92, 15.  ,
        9.01])
```

```
prediction_na[0:10]
```

```
array([[13.745984 ],
       [14.813007 ],
       [13.781742 ],
       [ 9.123977 ],
       [13.566464 ],
       [11.600856 ],
       [10.630115 ],
       [12.300923 ],
       [10.297885 ],
       [ 3.7286854]], dtype=float32)
```