Desenvolvimento de uma rede neural que maximiza os acertos se uma notícia irá ser popular

Alunas: Elaine Sangali, Ana Frozza

Introdução

O presente trabalho tem como objetivo projetar uma rede neural artificial que maximize os acertos de uma base de dados de notícias, onde o objetivo é prever se uma notícia obterá sucesso ou não. Para o desenvolvimento da rede será utilizado o Keras. Será testado vários métodos e valores de entradas para tentar obter o maior número de acerto possível. O conteúdo teórico deste trabalho pode ser encontrado no site do
[deeplearningbook.com.br](http://deeplearningbook.com.br/capitulos/).

Redes neurais

Uma rede neural tem como objetivo imitar como o cérebro humano aprende. Ela é um mecanismo de aprendizado de máquina muito poderoso, à medida que uma tarefa se torna complicada, há vários perceptrons que formam uma rede que transmitem informações entre si. Um perceptron representa um neurônio. O modelo do Perceptron foi desenvolvido nas décadas de 1950 e 1960 pelo cientista Frank Rosenblatt. Hoje é mais utilizado outros modelos de neurônnios artificias, mas esse seria um modelo básico, como mostra a figura 1, onde o perceptron rece várias entradas, x1; x2; x3 e produz uma única saída binária.

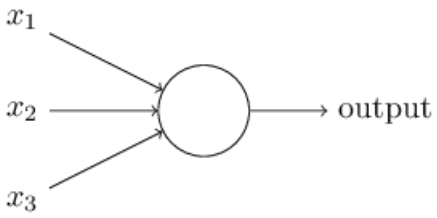


figura 1 - Modelo básico de um perceptron

No modelo da figura 1, o perceptron possui três entradas, x1; x2; x3, para calcular a saída, Rosenblatt introduziu pesos, w1; w2; w3, números reais que representam a importância das entradas para a saída, assim a entrada x1 possui peso w1, x2 peso w2 e x3 peso w3. A saída do neurônio é binária, 0 ou 1, e é determinada pela soma ponderada, Σjwjxj, menor ou maior do que algum valor limiar (threshold), como mostra a figura 2.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

figura 2 - Modelo algébrico da saída de um perceptron

O modelo da figura 1 seria um modelo básico de um perceptron, mas atualmente é utilizado modelos mais completos que obtem melhores resultados, como o modelo da figura 3. O modelo da figura 1, simplesmente utiliza uma somatória do produto dos pesos com as entradas, mas esse é um modelo muito simples para determinados problemas. No modelo da figura 3, a função de ativação g(.) usará a saída u em uma função, e o resultado do perceptron será a saída da função g. O simbolo Θ representa o viés (bias), que são utilizados no lugar do threshold, os bias são ajustadas da mesma forma que os pesos sinápticos, o bias permite que um neurônio apresente a saída não nula ainda que todas as suas entradas sejam nulas. O bias representa o quão fácil é fazer o perceptron produzir um 1 (disparar). Um perceptron com um viés muito grande tem uma tendência a emitir um 1, e muito pequeno de emitir 0.

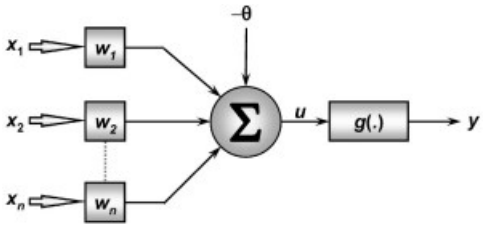


figura 3 - Modelo de perceptron com bia e função de ativação

O novo modelo utiliza uma função de soma um pouco diferente, ainda é realizado a soma dos produtos dos pesos com as entradas, mas no fim é somado o valor do viés, como mostra a figura 4.

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

figura 4 - Modelo algébrico com o viés

Um único perceptron não consegue resolver os problemas grandes, para isso é necessário uma rede de perceptrons. Há três categorias de tipos de redes de perceptrons (Arquiteturas):

1. Redes Neurais Feed-Forward: São mais comuns, a primeira camada é a entrada e a última camada é a saída, se houver uma camada oculta entre as duas, é chamado de redes neurais profundas(Deep Learning). A rede calcula uma série de transformação que altera a semelhança entre os casos, as atividades dos neurônios em cada camada são uma função não-linear das atividades na camada anterior.
2. Redes Recorrentes: Essa rede é utilizada quando para se obter o valor de saída atual é necessário analisar o valor do passado. Essa rede é equivalente a redes muito profundas com uma camada oculta por fatia de tempo; exceto que eles usam os mesmos pesos em cada fatia de tempo e recebem entrada em cada fatia. Eles têm a capacidade de lembrar informações em seu estado oculto por um longo período de tempo, mas é muito difícil treiná-las para usar esse potencial. Podem possuir uma dinâmica complicada, sendo difíceis de treinar, mas são mais biologicamente realistas.
3. Redes Conectadas Simetricamente: São como as redes recorrentes mas elas possuem o mesmo peso em ambas as direções. As redes conectadas simetricamente sem unidades ocultas são chamadas de “Redes Hopfield”. As redes conectadas simetricamente com unidades ocultas são chamadas de “Máquinas de Boltzmann”.

O trabalho atual se enquadra na categoria Redes Neurais Feed-Forward, a arquitetura utilizada é a Redes Multilayer Perceptrons (MLP), a rede MLP é composta por mais de um perceptron, e possui uma camada de entrada, uma de saída que toma uma decisão sobre a entrada, e entre as duas pode haver várias camadas ocultas. O MLP é muito utilizado em problemas de aprendizagem supervisionados, ele treina um conjunto de pares entrada-saída e aprende a modelar a correlação entre as entradas e saídas, no treinamento é realizado o ajuste dos parâmetros, pesos e bias da rede para conseguir minimizar o erro. O backpropagation é usado para fazer os ajustes dos pesos e de bias em relação ao erro, e o próprio erro pode ser medido de várias maneiras, inclusive pelo erro quadrático médio.

Base de dados: Online News Popularity

A base de dados Online News Popularity possui 60 atributos de várias notícias a ser analisados pela rede neural mais 1 atributo que possui o valor alvo, os atributos possuem o seguinte significado:

1. url: Url da notícia
2. timedelta: Dias entre a publicação da notícia e a aquisição do conjunto de dados (não-preditiva)
3. n_tokens_title: Quantidade de palavras do título
4. n_tokens_content: Quantidade de palavras do conteúdo
5. n_unique_tokens: Quantidade de palavras únicas no conteúdo
6. n_non_stop_words: Taxa de palavras sem parar no conteúdo
7. n_non_stop_unique_tokens: Quantidade de palavras não únicas no conteúdo
8. num_hrefs: Número de links
9. num_self_hrefs: Número de links para outras notícias publicados pela Mashable
10. num_imgs: Número de imagens
11. num_videos: Número de vídeos
12. average_token_length: Tamanho médio das palavras no conteúdo
13. num_keywords: Número de palavras-chave nos metadados
14. data_channel_is_lifestyle: É o canal de dados 'Lifestyle'?
15. data_channel_is_entertainment: O canal de dados é 'Entretenimento'?
16. data_channel_is_bus: É o canal de dados 'Business'?
17. data_channel_is_socmed: É o canal de dados 'Social Media'?
18. data_channel_is_tech: O canal de dados é 'Tech'?
19. data_channel_is_world: é o canal de dados 'World'?
20. kw_min_min: Pior palavra-chave (min. Compartilhamentos)
21. kw_max_min: Pior palavra-chave (máx. Compartilhamentos)
22. kw_avg_min: Pior palavra-chave (média de compartilhamentos)
23. kw_min_max: Melhor palavra-chave (min. Compartilhamentos)
24. kw_max_max: Melhor palavra-chave (máx. Compartilhamentos)
25. kw_avg_max: Melhor palavra-chave (média de compartilhamentos)
26. kw_min_avg: média palavra-chave (min. partes)
27. kw_max_avg: média palavra-chave (máx. compartilhamentos)
28. kw_avg_avg: média palavra-chave (média de compartilhamentos)
29. self_reference_min_shares: minimo de ações de notícias referenciados em Mashable
30. self_reference_max_shares: máx. ações de notícias referenciados em Mashable
31. self_reference_avg_sharess: média. ações de notícias referenciados em Mashable

32. weekday_is_monday: A notícia foi publicado na segunda-feira?
33. weekday_is_tuesday: A notícia foi publicado em uma terça-feira?
34. weekday_is_wednesday: A notícia foi publicado em uma quarta-feira?
35. weekday_is_thursday: A notícia foi publicado em uma quinta-feira?
36. weekday_is_friday: A notícia foi publicado em uma sexta-feira?
37. weekday_is_saturday: A notícia foi publicado em um sábado?
38. weekday_is_sunday: A notícia foi publicado em um domingo?
39. is_weekend: A notícia foi publicado no final de semana?
40. LDA_00: Proximidade do tópico 0 do LDA
41. LDA_01: Proximidade do tema 1 do LDA
42. LDA_02: Proximidade do tópico 2 do LDA
43. LDA_03: Proximidade do tema 3 do LDA
44. LDA_04: Proximidade do tema 4 do LDA
45. global_subjectivity: Subjetividade do texto
46. global_sentiment_polarity: polaridade do sentimento de texto
47. global_rate_positive_words: Taxa de palavras positivas no conteúdo
48. global_rate_negative_words: Taxa de palavras negativas no conteúdo
49. rate_positive_words: Taxa de palavras positivas entre tokens não neutros
50. rate_negative_words: Taxa de palavras negativas entre tokens não neutros
51. avg_positive_polarity: média polaridade de palavras positivas
52. min_positive_polarity: min. polaridade de palavras positivas
53. max_positive_polarity: máx. polaridade de palavras positivas
54. avg_negative_polarity: média polaridade de palavras negativas
55. min_negative_polarity: min. polaridade de palavras negativas
56. max_negative_polarity: máx. polaridade de palavras negativas
57. title_subjectivity: subjetividade do título
58. title_sentiment_polarity: polaridade do título
59. abs_title_subjectivity: Nível de subjetividade absoluta
60. abs_title_sentiment_polarity: nível de polaridade absoluta
61. ações: Número de ações (alvo)

Este conjunto de dados resume um conjunto heterogêneo de características sobre artigos publicados pela Mashable em um período de dois anos. O objetivo é prever o número de compartilhamentos nas redes sociais (popularidade). A base de dados contém 39644 exemplos. A média de compartilhamentos é de 3395, assim, valores abaixo dessa média serão considerados não populares, e valores iguais ou acima dessa média serão considerados populares.

Desenvolvimento

Inicialmente será importado as bibliotecas que serão utilizadas no projeto, e será utilizado um código simples do keras de classificação binária, pois o resultado é binário, ou é popular, ou não é.

Para separar a base em treino e teste foi utilizado o train_test_split como mostra a linha 21, para teste foi separado 30% da base. Além do teste e do treino, também foi separado 25% do treino para validação, afim de não ter uma base viciada.

In [2]:

```
#importando bibliotecas necessárias no projeto
from sklearn import svm
from keras.models import Sequential
from keras.utils import plot_model
from keras import regularizers, optimizers
from sklearn.model_selection import train_test_split
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.python.client import device_lib
from sklearn.svm import SVC
from keras import utils as np_utils
import numpy as np
import csv
import matplotlib.pyplot as plt
```

```
C:\Users\Elaine\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the seco
nd argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as
`np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [3]:

```
reader = csv.reader(open('OnlineNewsPopularity.csv','r'), delimiter=',') #lendo os atributos da base de
dados
```

```
dados

rows = np.array(list(reader))
labels = rows[0] #vetor com os labels das caracteristicas

X = rows[1:-1, 1:-1] #vetor de caracteristicas
Ya = rows[1:-1, -1] #Vetor de resultados

Y = []

index = 0

for y in Ya:
    if(int(y) >= 3395):
        Y.insert(index, True)
    else:
        Y.insert(index, False)

    index += 1

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, stratify=Y) #separando em um c
onjunto de treino e outro de teste

num_input = x_train.shape[1]
```

Função de ativação

A função de ativação utilizada foi a Relu, ela é a função de ativação mais utilizada em redes neurais hoje em dia. A função é definida como f(x) = max(0, x). A função não é linear, assim, se pode facilmente copiar os erros para trás e ter várias camadas de neurônios ativados pela função Relu. A sua principal vantagem é que ela não ativa todos os neurônios ao mesmo tempo, e a sua desvantagem é que ela pode ter problemas com os gradientes que se deslocam em direção a zero.

Método de descida de gradiente

O método de descida de gradiente utilizado foi o Adam. O Adam é um método para otimização estocástica eficiente que possui pouca exigência de memória, ele calcula as taxas individuais de aprendizagem adaptativa para diferentes parâmetros de estimativas de primeiro e segundo momentos dos gradientes. O método é simples de implementar, é computacionalmente eficiente, tem poucos requisitos de memória, é invariante para o reescalonamento diagonal dos gradientes e é bem adequado para problemas que são grandes em termos de dados e / ou parâmetros. O método também é apropriado para objetivos não estacionários e problemas com gradientes muito ruidosos e / ou esparsos.

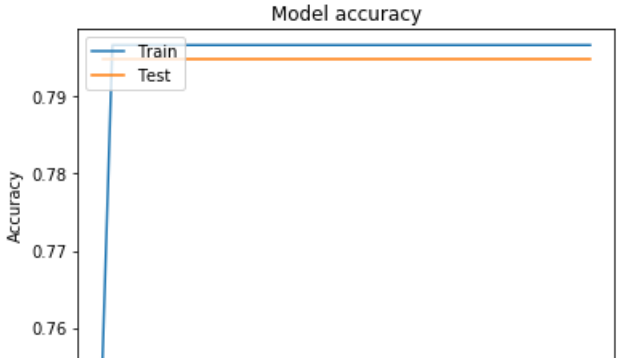
Função de custo

A cada iteração a rede neural precisa alterar os valores dos pesos para tentar chegar num resultado melhor, para isso é necessário uma função que nos mostre o quão boa é a solução atual, essa função é chamada de função de custo. A função de custo utilizada é a "binary_crossentropy", ela utiliza a medida de entropia cruzada, que é usada como uma medida de erro quando as saídas de uma rede podem ser pensadas como representando hipóteses independentes e as ativações dos nós podem ser entendidas como representando a probabilidade (ou a confiança) que cada uma das hipóteses pode ser verdadeira. A entropia cruzada indica a diferença do valor que é esperado e da saída real. A entropia cruzada é mais útil em problemas cujo objetivos são 0 ou 1.

Overfitting

O código inicial está representado a seguir demonstra o overfitting . O overfitting ocorre quando utilizamos neurônios igual ou maiores que 96 na segunda camada.

Como é observado na figura 6, o overfitting acontece devido ao fato de a rede neural apresentar melhores resultados no treino do que no teste.



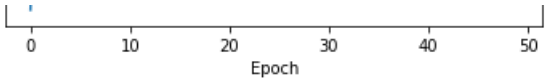


figura 6 - Resultado do overfitting

A figura 7 mostra a rede neural utilizando 90 neurônios, nessa fase não acontece o overfitting, pois o teste obtem melhores resultados que o treino.

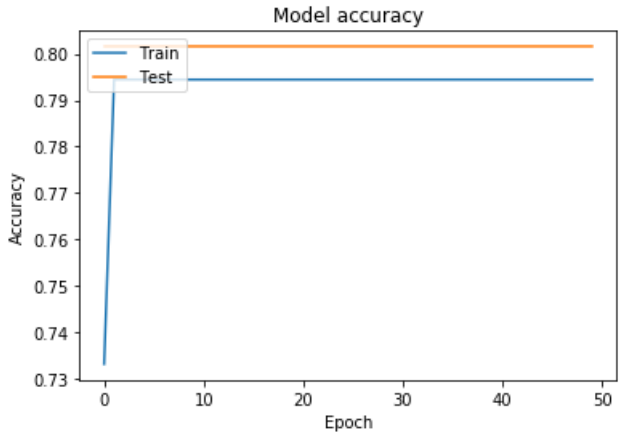


figura 7 - Resultado do overfitting

In [ ]:

```
model = Sequential()

model.add(Dense(units=100, activation='relu', input_dim=num_input))
model.add(Dense(units=59, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train, validation_split=0.25, epochs=50, batch_size=16, verbose=1)

loss_and_metrics = model.evaluate(x_test, y_test, batch_size=16)
print("\n Taxa de acerto: %.2f%%" % (loss_and_metrics[1]*100))

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
Train on 20812 samples, validate on 6938 samples
Epoch 1/50
```

```
Exception ignored in: <bound method BaseSession.__del__ of <tensorflow.python.client.session.Session ob
ject at 0x000001F3E41A15F8>>
Traceback (most recent call last):
  File "C:\Users\Elaine\Anaconda3\lib\site-packages\tensorflow\python\client\session.py", line 732, in
__del__
    try:
KeyboardInterrupt:
```

```
11936/20812 [================>.............] - ETA: 6s - loss: 3.6589 - acc: 0.7726
```

Regularização

A regularização pode ajudar a reduzir o overfitting. Analisaremos os efeitos de três técnicas de regularização no código, a técnica L1, L2 e Dropout. A intenção da regularização é fazer com que a rede prefira aprender pesos pequenos. Os pesos grandes só são permitidos se melhorarem bastante a primeira parte da função de custo, ou seja, ela tenta encontrar pequenos pesos e minimizar a função de custo original.

Tanto na técnica L1 quanto na L2 o resultado é a diminuição dos valores dos pesos, mas a maneira como os pesos diminuem é diferente. Quando um peso específico tem uma grande magnitude, a regularização L1 reduz o peso muito menos do que a Regularização L2, mas, quando |w| é pequeno, a regularização L1 reduz o peso muito mais do que a regularização L2, assim a regularização L1 tende a concentrar o peso da rede em um número relativamente pequeno de conexões de alta importância.

enquanto os outros pesos são direcionados para zero.

Configurações utilizadas no modelo de regularização L1:

In [ ]:

```
model = Sequential()

model.add(Dense(units=100, activation='relu', input_dim=num_input, activity_regularizer=regularizers.l1
(0.01)))
model.add(Dense(units=59, activation='relu', activity_regularizer=regularizers.l1(0.01)))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, validation_split=0.25, epochs=50, batch_size=16, verbose=1)

loss_and_metrics = model.evaluate(x_test, y_test, batch_size=16)

print("\n Taxa de acerto: %.2f%%" % (loss_and_metrics[1]*100))

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Com essas configurações no L1 foi obtido uma taxa de acerto de 79,63%. A figura 8 demostra o histórico da acurácia para o L1 com valor de 0.1.

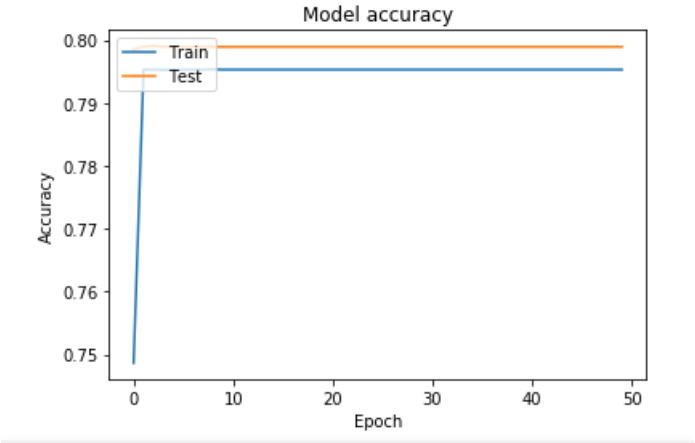


figura 8 - Histórico de acurácia para L1 = 0.1

A figura 9 demostra o histórico da acurácia para o L1 com valor de de 0.01 nesse caso o resultado do teste apresentou ser pior do que o utilizado com 0.1, pois a taxa de acerto obtida foi de 79,60%.

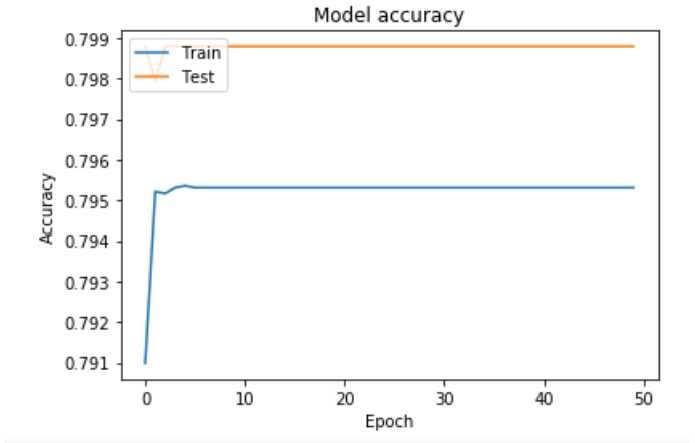


figura 9 - Histórico de acurácia para L1 = 0.01

Configurações utilizadas no modelo de regularização L2:

In [ ]:

```
model = Sequential()

model.add(Dense(units=100, activation='relu', input_dim=num_input, kernel_regularizer=regularizers.l2(0
.01)))
model.add(Dense(units=59, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, validation_split=0.25, epochs=50, batch_size=16, verbose=1)

loss_and_metrics = model.evaluate(x_test, y_test, batch_size=16)

print("\n Taxa de acerto: %.2f%%" % (loss_and_metrics[1]*100))

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Com essas configurações no L2 foi obtido uma taxa de acerto de 79,62%. A figura 10 demostra o histórico da acurácia para o L1 com valor de 0.01, a média da taxa de acerto de treino é de 79,61% e de teste é de 79,65%.

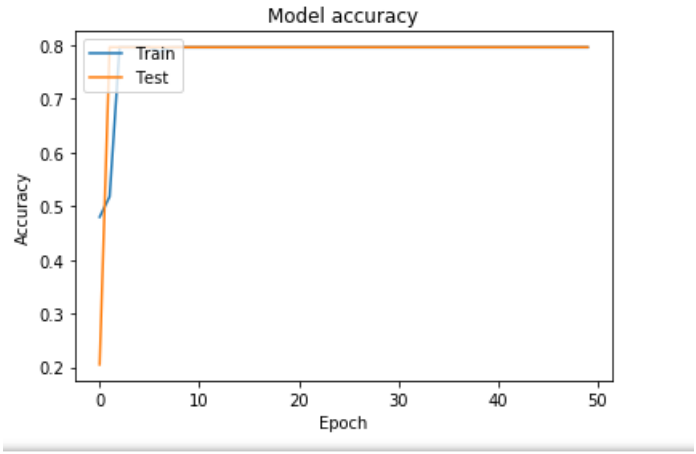


figura 10 - Histórico de acurácia para L2 = 0.01

A figura 11 demostra o histórico da acurácia para o L2 com valor de de 0.1, nesse caso o resultado do teste apresentou ser praticamente o mesmo que o anterior.

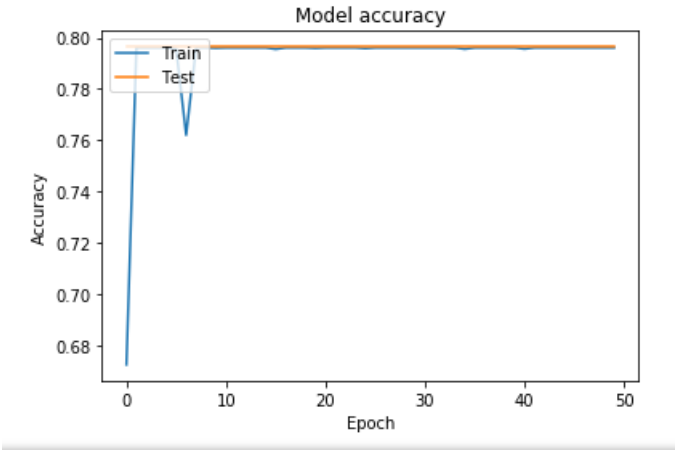


figura 11 - Histórico de acurácia para L2 = 0.1

O Dropout é uma técnica diferente da L1 e L2, ele não depende da modificação da função de custo, ele modifica a própria rede. O Dropout elimina aleatoriamente (e temporariamente) alguns dos neurônios ocultos na rede, mas deixa os neurônios de entrada e saída intocados.

Configurações utilizadas no Dropout:

In [ ]:

```
model = Sequential()

model.add(Dense(units=100, activation='relu', input_dim=num_input)
model.add(Dropout(0.5))
model.add(Dense(units=59, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, validation_split=0.25, epochs=50, batch_size=16, verbose=1)

loss_and_metrics = model.evaluate(x_test, y_test, batch_size=16)

print("\n Taxa de acerto: %.2f%%" % (loss_and_metrics[1]*100))

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Com o Dropout de 0.5 foi obtido uma taxa de acerto de 79,62%, a figura 12 mostra o histórico da acurácia na fase de treino e teste.

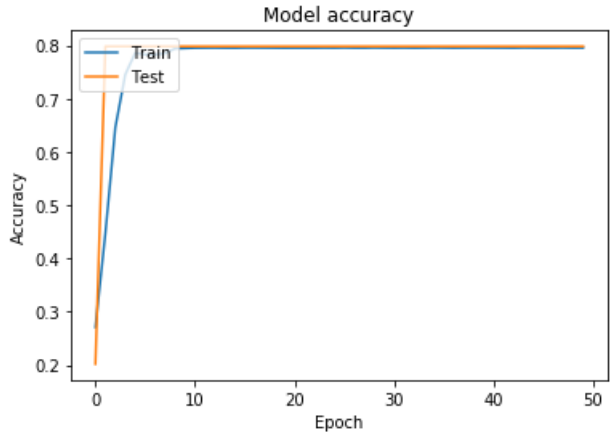


figura 12 - Histórico de acurácia para Dropout = 0.5

Com o Dropout de 0.3 foi obtido uma taxa de acerto de 79,62%, a figura 13 mostra o histórico da acurácia na fase de treino e teste.

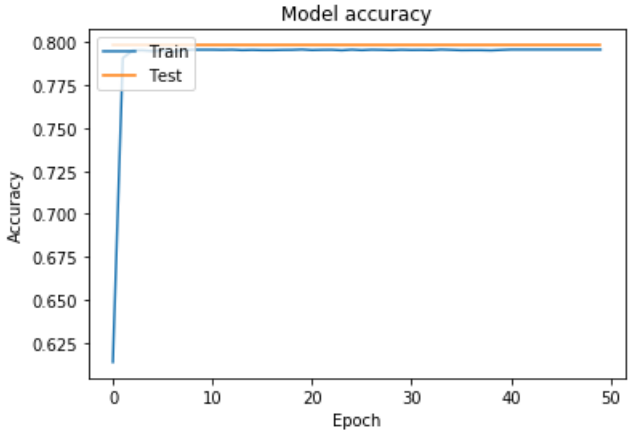
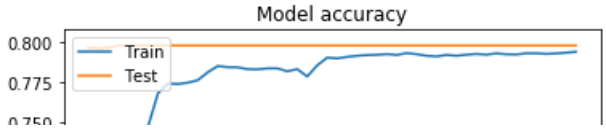


figura 13 - Histórico de acurácia para Dropout = 0.3

Com o Dropout de 0.7 foi obtido uma taxa de acerto de 79,62%, a figura 14 mostra o histórico da acurácia na fase de treino e teste.



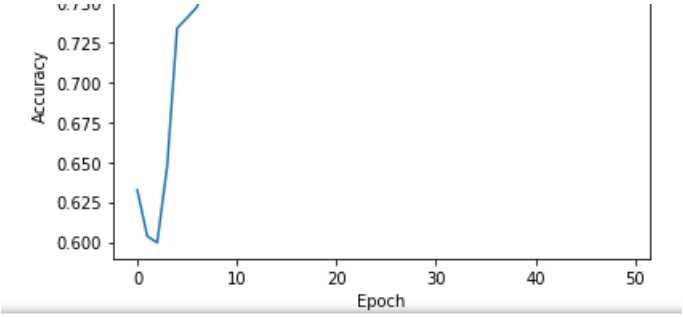


figura 14 - Histórico de acurácia para Dropout = 0.7

Hiperparâmetros

Inicialmente será decidido o número de neuronios utilizados na segunda camada oculta, como mostra o código a seguir:

In [ ]:

```
model = Sequential()

model.add(Dense(units=80, activation='relu', input_dim=num_input))
model.add(Dense(units=20, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_train, y_train, validation_split=0.25, epochs=5, batch_size=16, verbose=1)

print("\n Taxa de acerto: %.2f%%" % (loss_and_metrics[1]*100))

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

A figura 15 mostra o histórico da acurácia obtido, utilizando 20 neuronios na segunda camada oculta. A taxa de acerto foi de 20,37%.

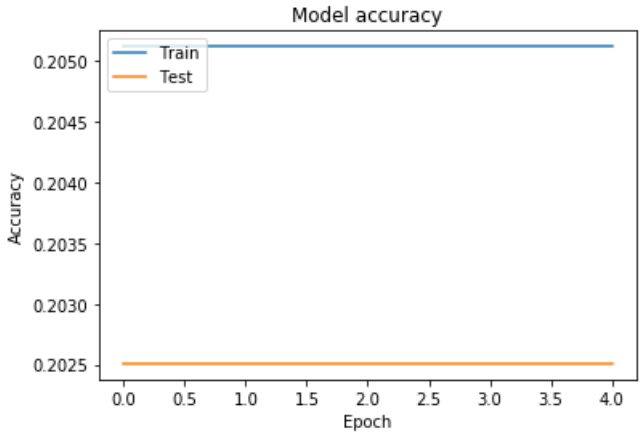
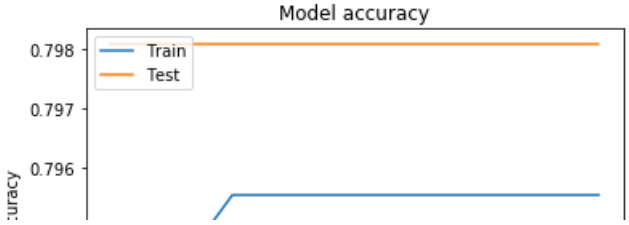


figura 15 - Histórico de acurácia para 2² camada oculta = 20

A figura 16 mostra o histórico da acurácia obtido, utilizando 59 neuronios na segunda camada oculta. A taxa de acerto foi de 79,62%. Foi testado até valores maiores, como 200 neuronios, mas apresentou o mesmo resultado, portanto a segunda camada oculta irá utilizar 59 neuronios.



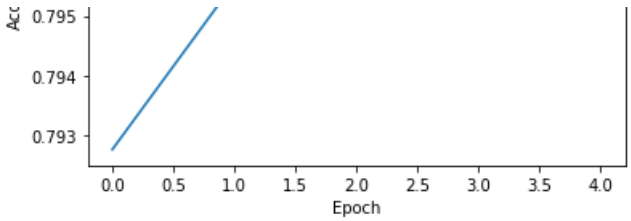


figura 16 - Histórico de acurácia para 2² camada oculta = 59

Como se pode observar, no primeiro caso ocorre o overfitting, então concluimos que valores baixos na segunda camada oculta não é bom, por isso será utilizado 59 neuronios.

Outro hiperparâmetro que pode ser ajustado é quantidade de dados retirados do treino para validação, foi testado utilizar valores de 25%, 30% e 40%, mas não houve variação na taxa de acerto, a mesma se manteve em 79.62% Então será mantido com o valor de 25%. Além da validação foi feito modificações no tamanho do batch, mas a medida que foi aumentando o valor do batch, o mesmo manteve a taxa de acerto, até cair para 20%.

Resultados

O melhor resultado que conseguimos obter foi o de 79,63%. Chegamos a esse resultado utilizando a regularização L1 na rede neural. A rede neural conseguiu obter bons resultados. De acordo com Kelwin Fernandes, utilizando 70% dos dados para treino e 30% para teste, o KNN e o Naive Bayes obteve uma taxa de acerto de 62%, o SVM e o AdaBoost de 66% e o RF de 67%. Para realizar mais comparações, também foi realizado teste para a base de dados utilizando a Árvore de Decisão, onde a mesma apresentou um resultado melhor, obtendo uma taxa de acerto de 79,66%.