Introdução e primeiros passos com Tensorflow no Python

Adilmar C. Dantas - Doutorando em Ciência da Computação

Universidade Federal de Uberlândia - Faculdade de Computação (FACOM)

10 de junho de 2019

Sumário

Introdução

Antes de entrarmos diretamente no assunto devemos ter bem estabelecidos alguns conceitos essenciais, um deles é *Deep-learning* ou (traduzido do inglês aprendizagem profunda). A aprendizagem profunda é um subcampo de aprendizado de máquina composta por um conjunto de algoritmos inspirados na estrutura e nas funcionalidades do cérebro.

Introdução

O TensorFlow é um framework de aprendizado de máquina que o Google criou e usou para projetar, construir e treinar modelos de aprendizagem profunda. Você pode usar a biblioteca TensorFlow para cálculos numéricos, o que por si só não parece muito especial, mas esses cálculos são feitos com grafos de fluxo de dados. Nesses grafos, os nós representam operações matemáticas, enquanto as arestas representam os dados, que geralmente são arrays ou tensores de dados multidimensionais, que são comunicados entre essas arestas. Logo o nome TensorFlow é derivado das operações que as redes neurais executam em matrizes ou tensores de dados multidimensionais!

Introdução - o que é um Tensor ?

Podemos definir um Tensor (T) como sendo uma estrutura de dados que contém valores primitivos e estão em um *array-n dimensional*.

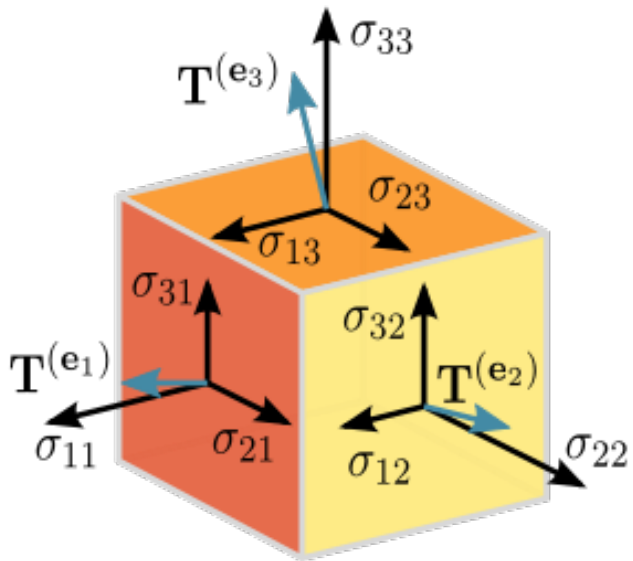


Figura: Representação de um Tensor com *n* dimensões.

Passo 1: instalando o Python

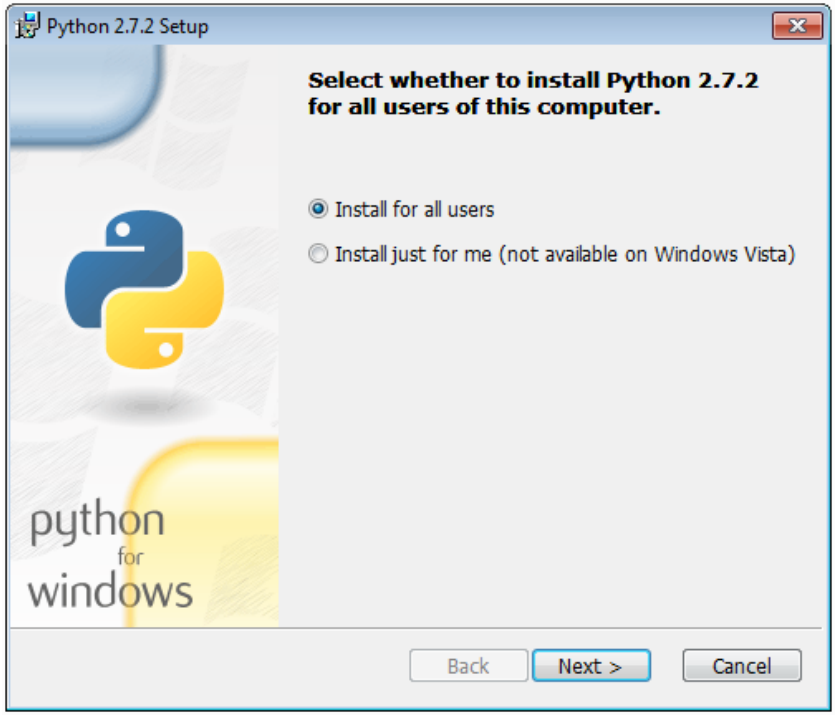


Figura: Instalação do Python

O primeiro passo a ser realizado é a instalação do Python versão 3.5. **Link Python - Windows X64**

Passo 2: instalando o Tensorflow

Para instalar o Tensoflow iremos utilizar o instalador de pacotes do próprio Python, executando os passos a seguir.

- Navegue pelo CMD até o diretório: **c:/Python3.5/Scripts/**
- Execute o comando **pip install tensorflow==1.12.0**
- Execute o comando **pip install keras**

Pronto o Tensorflow e suas dependências serão instaladas, agora vamos aos códigos.

Criando um Tensor (T)

Para usar o TensorFlow corretamente iremos utilizar os seguintes passos em nossos algoritmos:

- Importar o TensorFlow;
- Inicializar os tensores;
- Efetuar as operações desejadas;
- Exibir a saída das operações.

Criando um Tensor (T)

```
1  # Importando o tensorflow
2  import tensorflow as tf
3
4  # Inicializando as constantes
5  x1 = tf.constant([1,2,3,4])
6  x2 = tf.constant([5,6,7,8])
7
8  # Multiplicando os Tensors
9  result = tf.multiply(x1, x2)
10
11 # Print the result
12 print(result)
```

Figura: Criando o primeiro Tensor usando o TF.

Criando uma seção no TensorFlow

Agora que já sabemos como criar um tensor, vamos aprender a criar e utilizar uma seção *tf*. A seção *tf* serve para inicializar nossas constantes e garantir que elas não serão alteradas na execução,isso evita erros indesejados e economiza trabalho quando se tem várias operações com os tensores.

Criando uma seção no TensorFlow

```
1  # Import tensorflow
2  import tensorflow as tf
3
4  # Inicializando as constantes
5  x1 = tf.constant([1,2,3,4])
6  x2 = tf.constant([5,6,7,8])
7
8  # Multiplicando
9  result = tf.multiply(x1, x2)
10
11 # Inicializando seção
12 sess = tf.Session()
13
14 # Print do resultado
15 print(sess.run(result))
16
17 # Fechar a seção
18 sess.close()
```

Regressão linear

Antes de entrarmos a fundo nas redes neurais com TensorFlow vamos praticar um pouco usando Machine Learning por meio da regressão linear. Uma equação para se estimar a condicional (valor esperado) de uma variável y, dados os valores de algumas outras variáveis x. Pra essa prática vamos usar a base **brain-body.txt** que eu preparei para vocês.

Regressão linear

Essa base é composta pelo peso do cérebro e do corpo de vários animais. Nosso objetivo então é dado um peso x do cérebro de um animal realizar uma predição para saber o peso corporal (body) do animal por meio da regressão linear. Bom antes de começarmos vamos instalar algumas lib's que vamos precisar...

Regressão linear

- pip install sklearn
- pip install pandas
- pip install matplotlib

Além os seguintes passos para o exmperimento são esses:

- carregar as bibliotecas;
- ler nossa base txt;
- criar o modelo de treinamento (usando regressão linear);
- exibir o gráfico com os eixos x [Brain] e os eixos y [body]

**Lembre-se que os eixos y serão os resultados da predição com regressão linear.

Regressão linear

```
#import lib's
import pandas as pd
from sklearn import linear_model
import matplotlib.pyplot as plt

#read data
dataframe = pd.read_fwf('txt/brain_body.txt')
x_values = dataframe[['Brain']]
y_values = dataframe[['Body']]

#train model on data
body_reg = linear_model.LinearRegression()
body_reg.fit(x_values,y_values)

#visualize
plt.scatter(x_values,y_values)
plt.plot(x_values,body_reg.predict(x_values))
plt.show()
```

Figura: Regressão linear (x,y).

Regressão linear

Como resultado teremos um gráfico com os valores de x (pesos cerebrias) e os valores resultantes da predição y (pesos corporais).

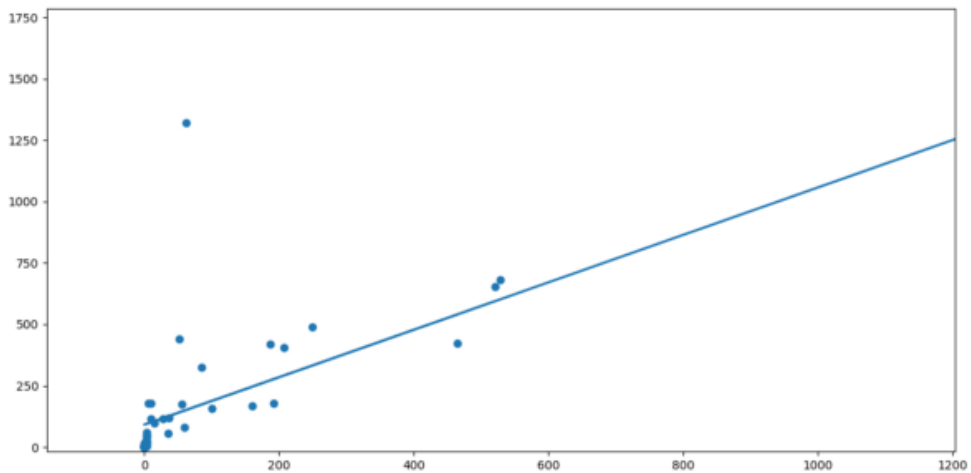


Figura: Gráfico resultante da regressão linear (x,y).

Regressão linear

Analisando o gráfico percebemos que todos os pontos foram mapeados corretamente e que a linha de regressão se adaptou muito bem, mas que não existe uma correlação forte que envolva o peso do cérebro com o peso corporal. Mas passando pela linha dado qualquer peso cerebral conseguimos predizer o peso corpo com a regressão linear.

Regressão linear

Modelos de regressão linear exibe o relacionamento entre uma variável independente e dependente para criar uma linha entre eles com o melhor ajuste possível onde podemos realizar nossas predições. Além disso antes de partimos para a implementação de redes neurais é necessário ter em mente que geralmente existem três meios de aprendizagem são eles:

Aprendizagem - supervisionada

Nesse modelo passamos um conjunto de dados rotulados, recebendo assim o feedback do que é e não é correto, tendo que aprender apenas o mapeamento entre os *labels* e os dados.

Figura: Exemplo de dados supervisionados (label: cat and food).

Aprendizagem - Não supervisionada

Nesse modelo passamos um conjunto de dados sem rótulos, além de não receber nenhum *feedback* sobre o que é ou não correto, tendo que aprender por si só, sendo o mais indicado uma vez que nem sempre temos a base bem formada com rótulos.

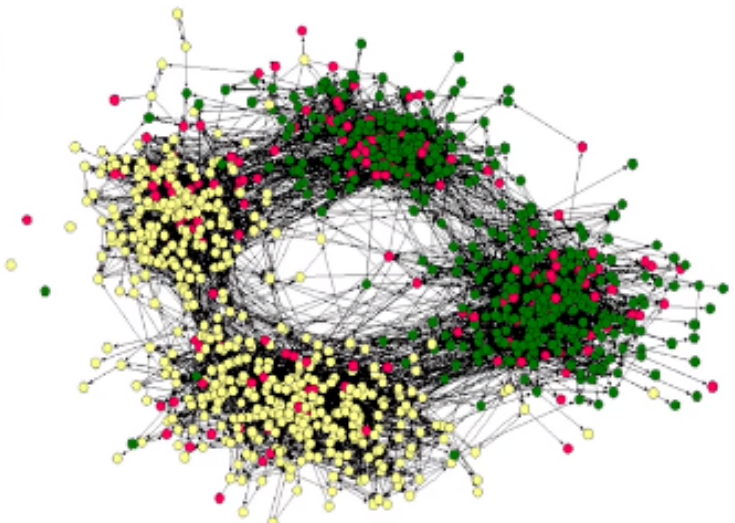


Figura: Exemplo de dados não supervisionados.

Aprendizagem por reforço

Nesse modelo o algoritmo se obtém o *feedback* apenas quando é alcançado o objetivo final. Ao contrario dos outros modelos a aprendizagem por reforço está interligada com a interação de tentativa e erro com o ambiente como em um jogo de xadrez.

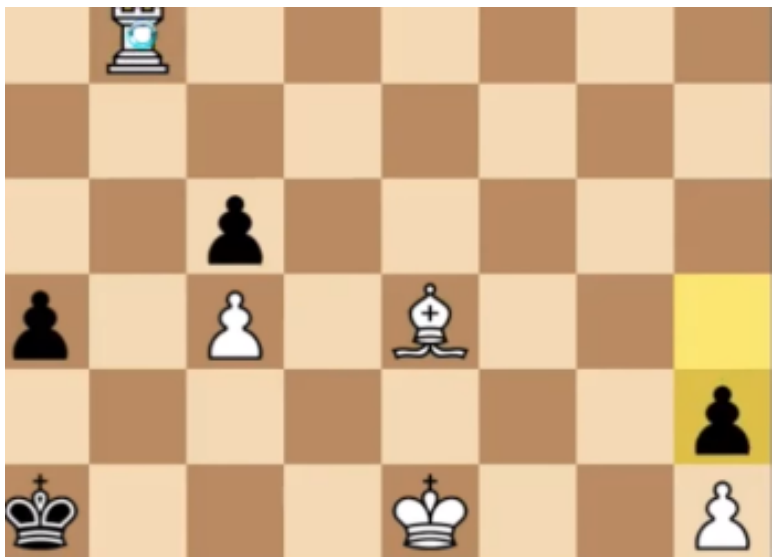


Figura: Xadrez uma exemplo de aprendizagem por reforço.

Redes neurais artificiais

Redes neurais artificiais (RNA) ou sistemas conexionistas são sistemas de computação inspirados pelas redes neurais biológicas. A rede neural em si não é um algoritmo, mas sim uma estrutura com muitos algoritmos de aprendizado de máquina diferentes com a finalidade de trabalharem juntos para processar entradas de dados complexas. Esses sistemas tem a capacidade de "aprender" a executar tarefas a partir de exemplos, geralmente sem serem programados com regras específicas [2].

Redes neurais artificiais

Por exemplo, no reconhecimento de imagem, elas podem **aprender a identificar** imagens que contenham gatos analisando imagens de exemplo que foram rotuladas manualmente como "gato" ou "cachorro", e usando os resultados para identificar gatos em outras imagens. Elas fazem isso **sem qualquer conhecimento prévio** sobre gatos, por exemplo, que eles têm peles, rabos, bigodes e faces de gato. Em vez disso, elas geram automaticamente características de identificação do material de aprendizagem que processam. Para o nosso problema utilizaremos um outro tipo de rede neural as **Feedforward Neural Network** a qual vamos explicar melhor a seguir.

Feedforward Neural Network

Diferente das redes neurais recorrentes (RNA), as redes neurais *feedforward* são um tipo de redes neurais em que as conexões entre os nós não formam um ciclo, como demonstrado na figura [2].

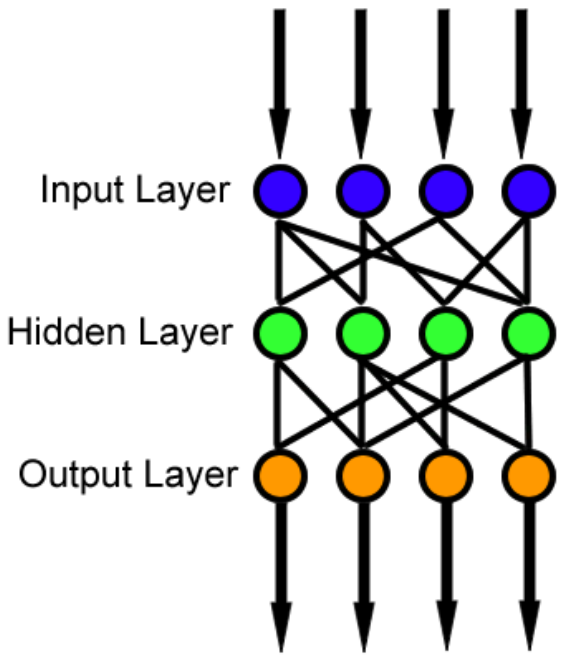


Figura: Rede neural Feedforward.

Feedforward Neural Network

Nesse tipo de rede, a informação move-se em apenas uma direção, para frente, dos nós de entrada, através dos nós ocultos (se houver) e para os nós de saída. Não há ciclos ou *loops* na rede. Agora que já conhecemos os principais tipos de redes neurais e seu funcionamento, podemos partir para o desenvolvimento de uma arquitetura para a resolução de problemas do cotidiano.

Escolhendo um problema

Para começarmos a se aventurar nessa nova área iremos pegar um problema clássico e resolve-lo utilizando redes neurais no Tensorflow. Vamos supor que para ir jogar tênis como os amigos tomamos como base informações do clima.

Outloook	Humidity	Wind	Play
Sunny	High	Weak	No
Sunny	High	Strong	No
Overcast	High	Weak	Yes
Rain	High	Weak	Yes
Rain	Normal	Weak	Yes
Rain	Normal	Strong	No
Overcast	Normal	Strong	Yes
Sunny	High	Weak	No
Sunny	Normal	Weak	Yes
Rain	Normal	Weak	Yes
Sunny	Normal	Strong	Yes
Overcast	High	Strong	Yes
Overcast	Normal	Weak	Yes
Rain	High	Strong	No

Jogar Tênis

Nesse conjunto de dados temos os registros de outlook, humidity, wind e play descrevendo se nessas condições ele jogou ou não tênis. Nosso objetivo e desenvolver uma rede que dado num novo conjunto de parâmetros (**outlook**, **humidity**, **wind**) seja capaz de predizer se iremos ou não jogar tênis.

Construção da rede neural - importação

O primeiro passo a ser realizado é a importação das bibliotecas necessárias:

- **import tensorflow as tf**
- **from tensorflow import keras**
- **import os**
- **import pandas**
- **import numpy as np**

Construção da Rede Neural - Arquitetura

Com as bibliotecas devidamente importadas o próximo passo é a definição da **arquitetura da rede** (parâmetros de entrada, saída, interações, épocas, etc). Logo em seguida deveremos criar uma função de treinamento para essa rede, para que seja possível passar uma nova informação para a realização da predição do conhecimento.

Construção da Rede Neural - Treinamento

Iremos começar criando uma função para receber nossos parâmetros para realizar o treinamento da rede.

```
1  def treinar():
2          #define model neural network
3          model = keras.Sequential()
4          #input layer - use tanh activation because job numbers 1 between -1
5          input_layer = keras.layers.Dense(3, input_shape=[3], activation='tanh')
6          model.add(input_layer)
7
8          #output layer use sigmoid because my output 0 or 1.
9          output_layer = keras.layers.Dense(1, activation='sigmoid')
10         model.add(output_layer)
11
12         # gradient of otimization train
13         gd = tf.train.GradientDescentOptimizer(0.01)
14
15         model.compile(optimizer=gd, loss='mse')
16
17         #open data-files
18         file = ("csv/play_tennis.csv")
```

Construção da Rede Neural - Treinamento

Agora iremos criar uma lógica para que o algoritmo seja capaz de ler o arquivo **Play-Tenis** com as informações iniciais para o treinamento além de converter os dados para o formato **np array** do Tensorflow.

```
def data_encode(file):
    X = []
    Y = []
    train_file = open(file, 'r')
    for line in train_file.read().strip().split('\n'):
        line = line.split(',')
        X.append([int(line[0]), int(line[1]), int(line[2])])
        Y.append(int(line[3]))
    return X, Y
```

Construção da Rede Neural - Treinamento

No final do processamento teremos como resultado dois Array's X (dados de entrada) e Y (dados de saída), com essas informações podemos criar a estrutura de treinamento e inicializar o processamento.

```
model.fit(training_x, training_y, epochs=8000, steps_per_epoch=10)

#save model use in the prediction
model.save('tennis.h5')

#test prediction
text_x = np.array([[1, 0, 0],[1,1,0]])

prediction = model.predict(text_x, verbose=0, steps=1)
print("predicao",prediction)
```

Construção da Rede Neural - Treinamento

Linha 1: Na função de treinamento model.fit() devemos passar como parâmetro as entradas (X), saídas (Y), o numero de épocas (número de vezes que as interações serão repetidas e por ultimo o numero de interações ou passos de cada época, para esse problema estamos usando 8000 (épocas) x 10 (interações).

Construção da Rede Neural - Treinamento

Linha 4: Logo em seguida optamos por salvar nosso modelo de treinamento usando a função model.save(), evitando assim um novo treinamento a cada nova predição.
Linha 9: Por fim realizamos uma predição para avaliar o modelo treinado e imprimimos o resultado da predição.

Construção da Rede Neural - Predição

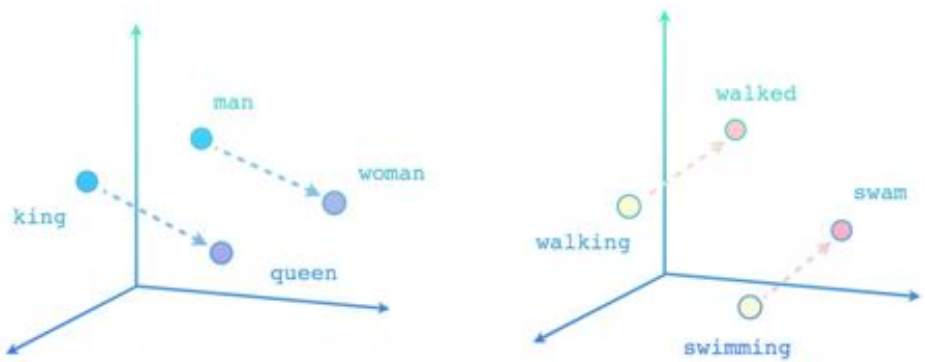
Agora que temos a estrutura da rede definida e já realizamos o treinamento da mesma, podemos realizar a predição para novas entradas. Para facilitar a nossa predição iremo criar uma função com essa funcionalidade, como principal característica iremos carregar o modelo salvo anteriormente na função de treinamento veja.

Construção da Rede Neural - Predição

```
def predicao(entrada):
        #load complete model
        model = keras.models.load_model('tennis.h5')

        #prin model resume
        #model.summary()

        #create input prediction
        text_x = np.array(entrada)

        #calcule prediction
        prediction = model.predict(text_x, verbose=0, steps=1)
        print("predicao",prediction)

#example
predicao([[1,0,1]])
```

Deep Learning - Representação da Linguagem

No aprendizado de maquinas existem duas maneiras de representar a linguagem: **junções vetoriais** ou **matrizes quentes**. As junções vetoriais são mapeamentos espaciais de palavras ou frases. A localização das palavras podem indicar semelhança e sugerir relações semânticas - por exemplo, incorporações vetoriais podem ser usadas para gerar analogias.



Deep Learning - Representação da Linguagem

As matrizes quentes, por outro lado, não contêm informação linguística. Esse tipo de abordagem é ingênuo; ou seja eles indicam quais dados contêm, mas nada sugerem sobre esses dados ou sua relação com outras informações. As matrizes quentes são chamadas de "quentes" porque cada uma delas incorpora uma dimensão de diferença uma da outra; cada matriz tem uma característica distintiva ("quente").

Deep Learning - Representação da Linguagem

Sendo assim podemos pegar todos os dados em nosso sistema e representá-los usando este sistema nivelado. Considere a seguinte frase "Eu adoro comer pizza com refrigerante", como podemos representar essa frase em uma matriz quente?. Uma boa estratégia seria **tokeninzar** a frase em palavras menores, veja:

```
Eu adoro comer pizza com refrigerante
```

```
['Eu', 'adoro', 'comer', 'pizza', 'com', 'refrigerante']
```

Deep Learning - Representação da Linguagem

Agora podemos criar um dicionário de pesquisa com a lista de palavras únicas. Agora, para criar as matrizes: cada *token* precisa ser transformado de uma string em uma matriz. Cada array é do tamanho do dicionário, e cada valor no dicionário que não é o valor do *token* atual é representado por um 0. O valor do *token* é representado por um 1.

```
{
    "eu":0,
    "adoro":1,
    "comer":2,
    "pizza":3,
    "com":4,
    "refrigerante":5
}
```

```
[1, 0, 0, 0, 0, 0], #eu
[0, 1, 0, 0, 0, 0], #adoro
[0, 0, 1, 0, 0, 0], #comer
[0, 0, 0, 1, 0, 0], #pizza
[0, 0, 0, 0, 1, 0], #com
[0, 0, 0, 0, 0, 1], #refrigerante
```

Deep Learning - Representação da Linguagem

- Agora podemos lidar com os *tokens* de maneira uniforme, já que todos eles são representados por estruturas de dados isomórficas e, depois que terminamos, podemos pesquisar seus valores usando o dicionário que fizemos anteriormente.
- As matrizes one-hot ficam grandes rapidamente. Nesse exemplo, usaremos apenas as 3000 palavras principais que ocorrem com mais frequência no corpus de treinamento. Isso significa que cada palavra é representada por uma matriz de 3000 itens.

Deep Learning - Construção da Rede

Redes neurais em geral demandam um grande tempo de treinamento, então no final do *script* de treinamento iremos salvar o modelo para realizarmos predições no futuro. Antes de começar vamos ver o roterio de treinamento:

- Baixar a base de dados - Tw Sentiment;
- Construir a rede neural;
- Treinar os dados;
- Salvar a rede para utilizações futuras.

Deep Learning - Organizando a base de dados

Estamos usando o conjunto de dados de análise de sentimentos do Twitter. Esteja preparado; esse conjunto de dados é extremamente grande e pode demorar uma eternidade para abrir no Excel :D, quando abrir teremos a seguinte estrutura.

ItemID	Sentiment	SentimentSource	SentimentText
1	0	Sentiment140	is so sad for my APL friend.....
2	0	Sentiment140	I missed the New Moon trailer...
3	1	Sentiment140	omg its already 7:30 :0

Deep Learning - Organizando a base de dados

As únicas colunas em que estamos interessados aqui são 1 e 3 - estaremos treinando nossa rede sobre entradas da coluna **Sentiment-Text** com saídas de **Sentiment**. Precisamos converter as postagens *SentimentText* em matrizes quentes e criar um dicionário com todas as palavras, para isso usaremos a biblioteca **numpy**.

Deep Learning - Organizando a base de dados

Vamos começar organizando nossos dados na estrutura do Tensorflow.

```
import json
import keras
import keras.preprocessing.text as kpt
from keras.preprocessing.text import Tokenizer
import numpy as np

# extract data from a csv
# notice the cool options to skip lines at the beginning
# and to only take data from certain columns
training = np.genfromtxt('data.csv', delimiter=',', skip_header=1, usecols=(1, 3), dtype=None, encoding='utf-8')

# create our training data from the tweets
train_x = [x[1] for x in training]
# index all the sentiment labels
train_y = np.asarray([x[0] for x in training])

# only work with the 3000 most popular words found in our dataset
max_words = 3000

# create a new Tokenizer
tokenizer = Tokenizer(num_words=max_words)
# feed our tweets to the Tokenizer
tokenizer.fit_on_texts(train_x)

# Tokenizers come with a convenient list of words and IDs
dictionary = tokenizer.word_index
# Let's save this out so we can use it later
with open('dictionary.json', 'w') as dictionary_file:
    json.dump(dictionary, dictionary_file)
```

Deep Learning - Construindo a Rede

```
def convert_text_to_index_array(text):
    # one really important thing that `text_to_word_sequence` does
    # is make all texts the same length -- in this case, the length
    # of the longest text in the set.
    return [dictionary[word] for word in kpt.text_to_word_sequence(text)]

allWordIndices = []
# for each tweet, change each token to its ID in the Tokenizer's word_index
for text in train_x:
    wordIndices = convert_text_to_index_array(text)
    allWordIndices.append(wordIndices)

# now we have a list of all tweets converted to index arrays.
# cast as an array for future usage.
allWordIndices = np.asarray(allWordIndices)

# create one-hot matrices out of the indexed tweets
train_x = tokenizer.sequences_to_matrix(allWordIndices, mode='binary')
# treat the labels as categories
train_y = keras.utils.to_categorical(train_y, 2)
```

Deep Learning - Construindo a Rede

Para construir nossa rede usaremos o Keras, ele torna o desenvolvimento de redes neurais o mais simples possível, com comendo de apenas uma linha veja como faremos:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation

model = Sequential()
model.add(Dense(512, input_shape=(max_words,), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='sigmoid'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

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

model.fit(train_x, train_y,
    batch_size=1,
    epochs=1,
    verbose=1,
    validation_split=0.1,
    shuffle=True)

model_json = model.to_json()
with open('model.json', 'w') as json_file:
    json_file.write(model_json)

model.save_weights('model.h5')

print('saved model!')
```

Deep Learning - Construindo a Rede

Parece bem simples né, mas como isso funciona ? Vamos la o **Sequential()** do Keras é um tipo simples de rede neural que consiste em uma "pilha"de camadas executadas em ordem. Se quiséssemos, poderíamos fazer uma pilha de apenas duas camadas (entrada e saída) para criar uma rede neural completa - sem camadas ocultas, mas assim não seria considerada uma rede neural profunda.

Deep Learning - Construindo a Rede

No nosso caso, temos como entrada uma matriz hot-length de tamanho max-words 3000. Incluímos também quantos outputs queremos sair dessa camada (512, para funsies) e o que tipo de maximização (ou "ativação") para usar. As funções de ativação são usadas ao treinar a rede; eles dizem à rede como julgar os pesos cada nó da rede criada. Nossa camada de saída consiste em duas saídas possíveis, já que são quantas categorias nossos dados podem ser classificados.

Deep Learning - Construindo a Rede

Entre as camadas de entrada e saída, temos mais uma camada Densa e duas camadas de Saída. Os *dropouts* são usados para remover dados aleatoriamente, o que pode ajudar a evitar o *overfitting*. O *overfitting* pode acontecer quando você continua treinando com dados iguais ou excessivamente parecidos - conforme você treina, sua precisão permanecerá firme ou diminuirá ao invés de aumentar.

Deep Learning - Construindo a Rede

Por último compilamos nosso modelo e especificamos que queremos coletar a métrica de precisão, isso nos dará uma saída ao vivo realmente útil à medida que treinamos nosso modelo.

```
1420764/1420764 [==============================] - 780s - loss:
0.4947 - acc: 0.7610 - val_loss: 0.4500 - val_acc: 0.7884
Epoch 2/5
1420764/1420764 [==============================] - 850s - loss:
0.4737 - acc: 0.7760 - val_loss: 0.4481 - val_acc: 0.7902
Epoch 3/5
1420764/1420764 [==============================] - 788s - loss:
0.4662 - acc: 0.7817 - val_loss: 0.4446 - val_acc: 0.7921
Epoch 4/5
1420764/1420764 [==============================] - 819s - loss:
0.4607 - acc: 0.7859 - val_loss: 0.4471 - val_acc: 0.7921
Epoch 5/5
1420764/1420764 [==============================] - 829s - loss:
0.4569 - acc: 0.7887 - val_loss: 0.4439 - val_acc: 0.7927
```

Deep Learning - Salvando nosso modelo

Algum tempo depois, vai depender do seu *hardware* teremos nosso modelo criado e treinado, você com certeza não quer treinar tudo novamente e além disso iremos precisar dele para a predição, para isso devemos salvar nosso modelo.

```
model_json = model.to_json()
with open('model.json', 'w') as json_file:
    json_file.write(model_json)

model.save_weights('model.h5')
```

Deep Learning - Construindo a Predição

Finalmente podemos construir nosso algoritmo para carregar nosso modelo e realizar as predições das frases.

```
import json
import numpy as np
import keras
import keras.preprocessing.text as kpt
from keras.preprocessing.text import Tokenizer
from keras.models import model_from_json

# we're still going to use a Tokenizer here, but we don't need to fit it
tokenizer = Tokenizer(num_words=3000)
# for human-friendly printing
labels = ['negativo', 'positivo']

# read in our saved dictionary
with open('dictionary.json', 'r') as dictionary_file:
    dictionary = json.load(dictionary_file)

# this utility makes sure that all the words in your input
# are registered in the dictionary
# before trying to turn them into a matrix.
def convert_text_to_index_array(text):
    words = kpt.text_to_word_sequence(text)
    wordIndices = []
    for word in words:
        if word in dictionary:
            wordIndices.append(dictionary[word])
        else:
            print("'%s' not in training corpus; ignoring." %(word))
    return wordIndices

# read in your saved model structure
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
# and create a model from that
model = model_from_json(loaded_model_json)
# and weight your nodes with your saved values
model.load_weights('model.h5')
```

Deep Learning - Construindo a Predição


```
def avaliar(evalSentence):
    if len(evalSentence) == 0:
        return
    # format your input for the neural net
    testArr = convert_text_to_index_array(evalSentence)
    input = tokenizer.sequences_to_matrix([testArr], mode='binary')
    # predict which bucket your input belongs in
    pred = model.predict(input)
    # and print it for the humons
    print("%s sentimento; %f%% confianca" % (labels[np.argmax(pred)], pred[0][np.argmax(pred)]
```

Deep Learning - Construindo a predição

Obrigado pessoal e continuem seus estudos em Tensorflow e analise de dados é um caminho sem volta rsrsrs ;D [1].

Referencias I

Google.
An end-to-end open source machine learning platform
@ONLINE, jan 2019.

S. Russell and P. Norvig.
Inteligência artificial.
CAMPUS - RJ, 2004.

Contatos e social