LEVANDO IA PARA PRODUÇÃO

UNIDADE IV
PRODUÇÃO PARA APLICAÇÕES QUE UTILIZAM INTELIGÊNCIA ARTIFICIAL

Elaboração

Paulo Vitor Pereira Cotta

Natasha Sophie Pereira

Produção

Equipe Técnica de Avaliação, Revisão Linguística e Editoração

SUMÁRIO

PRODUÇÃO PARA APLICAÇÕES QUE UTILIZAM INTELIGÊNCIA ARTIFICIAL

UNIDADE IV

CAPÍTULO 1

IA PARA PRODUÇÃO

Inteligência Artificial para Produção

Com o surgimento da indústria 4.0 e das novas tecnologias, não há mais espaço para estender o tempo na construção de sistemas para irem do ambiente de teste para o de produção. Cada vez mais as grandes indústrias estão investindo na automatização da fabricação de seus produtos, pois o usuário final necessita de comodidade.

Nesse contexto, a inteligência artificial assume um papel central: a facilitação com a automatização de toda a cadeia produtiva na organização e/ou empresa. Uma série de impactos é esperada sobre a nova indústria, com criação de novos empregos e melhoria na qualidade de vida.

Existem diversas aplicações da inteligência artificial dentro das organizações como um todo que não estão ligadas diretamente apenas à produção, mas também à administração da empresa e à logística, marketing, entre outros, conforme apresentado abaixo:

Prever comportamentos dos usuários

Esses sistemas podem ser utilizados em um formato estratégico para prever/analisar comportamentos relacionados aos seus usuários, como preferências e demandas. Mas nem todas as empresas, ou a própria área de TI, estão preparadas, o que faz com que devamos aprender mais sobre IA.

Os *softwares* de IA podem construir modelos de clientes em potencial e avaliar prospecções que podem ser utilizadas para criar campanhas personalizadas e atingir o usuário da melhor forma possível e com custos mais baixos.

Sistemas de IA também podem ser utilizados para prever comportamentos de mercado, marketing direcionado, identificando riscos ou oportunidades para a manufatura, como

queda ou aumento de demanda devido a uma variável específica dentro da predição dos dados.

Auxiliar na tomada de decisões estratégicas da empresa

Tomar decisões de forma rápida e da melhor maneira possível tem sido um dos grandes desafios das empresas e das grandes organizações. O processo é tranquilo de entender. Com o processo globalizado e a facilidade de importação e exportação entre os países, a competição nos mercados em esfera mundial aumentou significativamente e é preciso se adequar a tempo para obter e ou reter informações.

No cenário com uma grande gama de empresas e ou fornecedores de IA, uma decisão errada pode fazer com que a sua empresa perca espaço no mercado, enfrente muitas dificuldades para se recuperar ou entre em processo de falência por decisões tomadas pela IA de forma errada.

Nesse sentido, é possível utilizar *softwares* de IA para extrair dados e buscar por pensamentos temporais e informações em que é relevante que colaborem para efetuar tomadas de decisões rápidas e muito mais seguras dentro de um curto prazo.

Automatizar a produção com IA

O principal objetivo das indústrias que buscam sistemas de IA é a automatização de processos, tomadas de decisões, ou de uma parte deles. Hoje, existem diversas ferramentas no mercado que podem auxiliar nessa questão.

O uso habitual é o investimento na robotização de parte da produção ou *software* com IA. A utilização de robôs é mais comum na indústria automobilística, que utiliza esse recurso nas linhas de produção, e o uso de *softwares* com IA é comum em empresas de mídias sociais ou redes sociais.

Contudo, com as técnicas e elaboração cada vez mais aprimoradas de *Machine Learning* e *Deep Learning*, IoTl, cada vez mais fábricas utilizarão robôs nos seus processos de produção, pois o custo fica mais barato e mais eficiente.

Minimizar os custos de produção com IA

Uma das aplicações indiretas da IA está relacionada à minimização dos custos diretos com os serviços ofertados pelas empresas. Isso ocorre pela diminuição de colaboradores, pela adaptação da produção de acordo com a demanda, do mapeamento de rotas a partir dos dados disponibilizados (clima, tráfego/trânsito, entre outros) e do controle

de estoques de forma eficaz, evitando desperdício. De modo geral, os sistemas de IA colaboram para a diminuição dos gastos operacionais das empresas.

Existem diversos custos dentro de uma empresa ou na indústria que acabam por passar despercebidos aos olhos dos gestores, donos, funcionários, mas que podem ser identificados e evitados por sistemas de IA.

A IA já é uma realidade em diversas indústrias, e a tendência é que ela se torne cada vez mais comum nas empresas, indústrias, residências e na vida das pessoas. Para manter a competitividade, é preciso estar trabalhando com esse tema desde já para buscar soluções que preparem a empresa para implantar sistemas de IA.

Como funciona o processo nas empresas com inteligência artificial?

Nas empresas que são orientadas em Data Driven, o mais importante para a empresa são os dados, e em empresas que não são Data Driven é mais complicado aplicar IA, já que os dados não estão normalizados e muitos não conhecem nem os dados que possuem em suas bases de dados.

Então vamos aos detalhes de como fazer para que os sistemas possam ir para produção em ambas as situações.

A ideia de Data Driven ou empresa orientada a dados diz respeito a uma solução que poderá trazer respostas a questões fundamentais para as decisões que ditarão os rumos de um negócio, o que é de grande importância para as empresas. Algumas questões que podem ser respondidas mais facilmente com as informações disponíveis são:

- Qual cliente tem mais propensão a consumir o produto ou serviço?
- Em qual momento minha abordagem ao cliente será mais assertiva?
- Qual é o perfil dos usuários da minha campanha de marketing?
- Quais fatores tornam a concorrência ou *startups* uma ameaça ao meu negócio?
- Quais parceiros são confiáveis ou imprescindíveis para trazer para minha estratégia empresarial?
- Qual é o tamanho do risco de se oferecer crédito a determinado perfil de cliente?

Um dos conceitos mais comuns de *data driven* é que se trata de uma metodologia orientada a dados do início ao fim, que podem ser de pessoas físicas ou jurídicas, na condição de clientes, parceiros, fornecedores ou concorrentes, entre outros. Essas

soluções se baseiam em algoritmos que cruzam grandes volumes desses dados ou bases de terceiros e os transformam em respostas para o sucesso do negócio.

Como benefício, esse método permite que as empresas errem menos em suas estratégias ou modelos de negócio. Isso porque são oferecidos elementos fundamentais para decisões e abordagens mais acertadas, que dão a real noção de tempo, direção e esforço que serão necessários para que determinada estratégia dê resultados.

As profissões que as empresas *Data Driven* procuram são:

- engenheiro de *Big Data*;
- cientista de dados;
- engenheiro de *Machine Learning*;
- engenheiro de *backend*;
- engenheiro de *frontend*;
- estatístico;
- engenheiro de dados e negócios.

Seguem os pontos de como uma empresa *Data Driven* coloca uma IA em produção:

- como primeiro ponto, os engenheiros de *Big Data* devem montar todo o ambiente de *Big Data* conforme apresentado na Unidade II;
- após isso, adicionar todo os dados dentro do cluster de Big Data;
- o cientista de dados analisa e separa os dados de acordo com o que foi levantado pelo engenheiro de dados e negócios;
- o engenheiro de *Machine Learning* recebe os dados e executa uma análise dos dados para identificar os modelos de *Machine Learning* ou *Deep Learning* que devem ser feitos;
- o estatístico, muitas vezes, é acionado para avaliar os dados e elencar as melhores variáveis. Dessa forma, avalia o modelo juntamente com o engenheiro de *Machine Learning*;
- em seguida, deve-se executar o treinamento do modelo e salvar seus pesos, que são os dados que foram aprendidos;
- o desenvolvedor *backend* recebe os pesos e utiliza para subir os dados dentro do seu código fonte os pesos gerados pelo modelo;

- após isso, o desenvolvedor *frontend* recebe os *endpoints* do *microservice* para inserir nas telas de acesso;
- muitas vezes, o desenvolvedor repassa para os ambientes um Docker;
- esse Docker disponibiliza o mesmo código para os ambientes de Desenvolvimento, Homologação e Produção;
- de tempos em tempos, os engenheiros devem avaliar a qualidade do modelo, pois há perda de qualidade nos processos de aprendizado.

Conforme apresentado, o processo de ir para produção com uma empresa com segmentos para Data Driven é muito satisfatório e mais fácil, pois cada processo fica bem desenhado e uniforme com o seu pipeline.

Para empresas que não são Data Driven, o processo é um pouco mais oneroso e complicado. Mas é possível, desde que a organização queira IA em sua tecnologia e siga alguns ajustes:

- os responsáveis da empresa devem entender que é um processo evolutivo;
- os dados devem ser avaliados para tomada de decisão;
- os dados devem ser organizados e distribuídos para as demandas;
- os funcionários da tecnologia da informação devem entender a ideia e ajudar no processo evolutivo que a empresa deseja exercer;
- e o aculturamento do *microservices*.

Esse processo não ocorre em muitas empresas que não sabem como cuidar dos dados e querem inteligência artificial. No final, acabam gastando muito dinheiro, ou até mesmo falindo, por decisões erradas na tomada de decisão. Além, é claro, de não detalhar e entender o papel de quem deve exercer cada cargo. Exemplo disso é o fato de empresas acharem que, para ter inteligência artificial, devem-se contratar pessoas com uma menor capacidade sobre o tema, ou até mesmo não conhecer do assunto e contratar profissionais que não dominam a área.

Inteligência Artificial, por sua vez, é uma área bem complexa e extensa.

Para empresas que não são *Data Driven*, devemos alertar aos perigos:

- dados com levantamento de forma errada podem ocasionar perda de dinheiro;
- frustração com a tecnologia;
- perda de tempo e, consequentemente, entendimento errado da IA;

» caso a empresa queira utilizar recursos em excesso, com uma predição feita de forma errada, pode ir à falência.

Como uma empresa que não é *Data Driven* pode ter IA? Ela deve se organizar de forma consistente e quebrar as áreas que devem e querem participar no processo de construir uma ou várias IAs.

A área mais importante para essa organização é a curadoria dos dados, já que estes estão desorganizados. Devem ser avaliados os dados, e deve-se executar a tentativa máxima de rotular/definir o que cada um daqueles dados quer dizer.

Além da curadoria dos dados, a existência de profissionais competentes desempenhando os cargos definidos para empresas *Data Driven* também é importante.

Caso a empresa não defina que deve ter esses cargos, por questões financeiras e/ou qualquer outro ponto, pode contratar serviços prontos e pessoas qualificadas na ferramenta paga, ou contratar pessoas para assumir mais de uma função (o que não é a melhor estratégia).

O processo para ir para produção deve seguir a melhor estratégia que a empresa tiver disponível no momento, mas seguir os passos definidos por uma empresa *Data Driven* é a melhor solução para não errar.

Conforme demonstrado na imagem, as APIs REST ficam expostas, enquanto são liberadas e consumidas dentro do modelo, e se utiliza a solução Hadoop como armazenamento. O interessante é que todos os itens que foram apresentados são o*pen source*, para baratear o custo e facilitar que as empresas tenham IA em seus domínios.

Após esse entendimento, colocar aplicações utilizando IA nas empresas é relativamente fácil, e o complicador é ter pessoas qualificados sobre o tema.

Como fazer uma aplicação que utiliza IA?

Nesse processo, deve-se ter uma boa lógica de programação e conhecimentos dos fundamentos de IA.

Vamos iniciar com um projeto *Flask Python* para os serviços REST. O que é o *Flask*? *Flask* é um *framework* web/WSGI escrito em Python e em C, baseado na biblioteca WSGI Werkzeug e na biblioteca de Jinja2. A licença é sob os termos protegidos BSD.

O *Flask* tem a total flexibilidade da linguagem de programação Python e provê um modelo simples para desenvolvimento web e APIs. Uma vez importado no Python, *Flask* pode

ser usado para economizar tempo construindo aplicações web, facilita no aprendizado, já que é um *framework* simples, e tem muitos exemplos disponíveis.

É chamado de *microframework* ou pequeno *framework* porque mantém um núcleo simples, mas estendível para encaixar mais *frameworks*. Não há uma camada de abstração do banco de dados como o DJango, validação de formulários ou qualquer outro componente criado em que bibliotecas de terceiros existem para prover a funcionalidades internas e externas. Assim, *Flask* suporta extensões capazes de adicionar tais funcionalidades na aplicação final, no que dá maior controle e liberdade para o desenvolvedor. Há uma vasta coleção de bibliotecas para resolver essas questões em Python. Isso simplifica o *framework* e torna sua curva de aprendizado mais suave.

Flask é um *microframework* desenvolvido em Python e baseado em 3 pilares:

» WerkZeug é uma biblioteca para desenvolvimento de aplicativos WSGI, que é a especificação universal Python de como deve ser a interface entre uma aplicação Python e um web server. Possui a implementação básica desse padrão para interceptar requests e lidar com response, controle de *cache*, *cookies*, *status* HTTP 1.1, roteamento de URLs e também conta com uma poderosa ferramenta de *debug* para facilitar encontrar erros. Além disso, o WerkZeug possui um conjunto de bibliotecas que acabam facilitando a construção de projetos que não são para a web;

» Jinja2 é uma ferramenta de execução escrita em Python. Você escreve htmls utilizando marcações como {{nome_da_variavel}} ou {%for alguma_coisa in lista_alguma_coisa%} ou Hello {{nome}}!! {%endfor%}, e o Jinja2 se encarrega de renderizar as páginas, ou seja, ele substitui as tags pelo valor de suas variáveis. O Jinja2 já vem com a implementação da maioria dos recursos necessários na construção de templates HTML e, além disso, é muito fácil de ser customizado com filtros de páginas etc.;

» *Good Intentions*: além de o código ter alta qualidade nos quesitos de legibilidade do código, *clean code*, ele também tenta seguir as premissas do *framework* Zen do Python2 e, dentro dessas boas intenções, nós temos o fato de ele ser um *miniframework*, deixando que você tenha liberdade de estruturar a aplicação da maneira que desejar. E, além de tudo disso, a comunidade é bastante ativa e compartilha muitos projetos de extensões *open source*, como o *Flask Admin*, *Flask-Cache, Flask-Google-Maps, Google-Flask, Flask-Mongoengine, Flask-SQLAlchemy, Flask-Login, Flask-Mail*, entre outros.

Mas, antes, é necessário instalar o Python e outras dependências, a saber:

» instalar o Python 3.7;

- um editor de código ou IDE de sua preferência (Eclipse, PyCharm, Gedit, Notepad++, Sublime, Emacs, VIM etc.);
- instalar o *Flask*.

Segue exemplo de código em *Flask*:

```
import logging
from flask import Flask
from flask import request
from flask_restplus import Resource, Api
from sauron_telegram.api.restplus import api
from sauron_telegram.api.endpoints.bot_sauron import send_message
from sauron_telegram.api.endpoints.bot_sauron import send_message_image
from sauron_telegram.api.endpoints.bot_sauron import transform_image
log = logging.getLogger(__name__)
ns = api.namespace('endpoints/posts', description='Post operação.')
TOKEN = "637217321:AAFqSw0KJnwpbhm3upF4qcCAHpxjPn6n_fQ"
URL = "https://api.telegram.org/bot{}/".format(TOKEN)
@ns.route('/')
class PostsCollection(Resource):
    def get (self):
        return "teste"
    @api.response ( 200, 'Enviado para o bot Sauron.' )
    def post (self):
        objSauron = request.json
        send_message(objSauron['message'], objSauron['chat_id'])
                        send_message_image(objSauron['chat_id'],   transform_
image(objSauron['photo']))
        return None , 200
```

Conforme apresentado na imagem acima, temos duas requisições, uma que chama via GET e outra via POST. Na Unidade III, apresentamos detalhes sobre as requisições REST.

Após ter o projeto *Flask* rodando, vamos agora ver um projeto com o processo de modelo de *Machine Learning* e *Deep Learning*. Todos os códigos foram implementados pelo desenvolvedor Paulo Vitor Pereira Cotta e estão disponibilizados no GtiHub: https://github.com/paulovpcotta.

```
import os
from PIL import Image
import numpy as np
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import AveragePooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.models import model_from_json
from keras.preprocessing.image import ImageDataGenerator
from scipy.ndimage import imread
```

```
from scipy.misc import imresize, imsave
IMG_SIZE = 24
def collect():
      train_datagen = ImageDataGenerator(
                  rescale=1./255,
                  shear_range=0.2,
                  horizontal_flip=True,
            )
      val_datagen = ImageDataGenerator(
                  rescale=1./255,
                  shear_range=0.2,
                  horizontal_flip=True,          )
      train_generator = train_datagen.flow_from_directory(
          directory="dataset/train",
          target_size=(IMG_SIZE, IMG_SIZE),
          color_mode="grayscale",
          batch_size=32,
          class_mode="binary",
          shuffle=True,
          seed=42
      )
      val_generator = val_datagen.flow_from_directory(
          directory="dataset/val",
          target_size=(IMG_SIZE, IMG_SIZE),
          color_mode="grayscale",
          batch_size=32,
          class_mode="binary",
          shuffle=True,
          seed=42
      )
      return train_generator, val_generator

def save_model(model):
      model_json = model.to_json()
      with open("model.json", "w") as json_file:
            json_file.write(model_json)
      # serialize weights to HDF5
      model.save_weights("model.h5")
def load_model():
      json_file = open('model.json', 'r')
      loaded_model_json = json_file.read()
      json_file.close()
      loaded_model = model_from_json(loaded_model_json)
      # load weights into new model
      loaded_model.load_weights("model.h5")
      loaded_model.compile(loss='binary_crossentropy',     optimizer='adam',
metrics=['accuracy'])
      return loaded_model
def train(train_generator, val_generator):
      STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
      STEP_SIZE_VALID=val_generator.n//val_generator.batch_size
      print('[LOG] Intialize Neural Network')
```

```
      model = Sequential()
      model.add(Conv2D(filters=6,  kernel_size=(3,  3),  activation='relu',
input_shape=(IMG_SIZE,IMG_SIZE,1)))
      model.add(AveragePooling2D())
      model.add(Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
      model.add(AveragePooling2D())
      model.add(Flatten())
      model.add(Dense(units=120, activation='relu'))
      model.add(Dense(units=84, activation='relu'))
      model.add(Dense(units=1, activation = 'sigmoid'))

      model.compile(loss='binary_crossentropy',           optimizer='adam',
metrics=['accuracy'])
      model.fit_generator(generator=train_generator,
                          steps_per_epoch=STEP_SIZE_TRAIN,
                          validation_data=val_generator,
                          validation_steps=STEP_SIZE_VALID,
                          epochs=20
      )
      save_model(model)
def predict(img, model):
      img = Image.fromarray(img, 'RGB').convert('L')
      img = imresize(img, (IMG_SIZE,IMG_SIZE)).astype('float32')
      img /= 255
      img = img.reshape(1,IMG_SIZE,IMG_SIZE,1)
      prediction = model.predict(img)
      if prediction < 0.1:
            prediction = 'closed'
      elif prediction > 0.9:
            prediction = 'open'
      else:
            prediction = 'idk'
      return prediction
def evaluate(X_test, y_test):
      model = load_model()
      print('Evaluate model')
      loss, acc = model.evaluate(X_test, y_test, verbose = 0)
      print(acc * 100)
if __name__ == '__main__':
      train_generator , val_generator = collect()
      train(train_generator,val_generator)
```

O código acima recebe os *frames* (imagens) de um vídeo para processar de acordo com a parte do treinamento. Então ele estabiliza a imagem e a deixa de acordo com o esperado para entrar diretamente na rede neural.

```
def train(train_generator, val_generator):
      STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
      STEP_SIZE_VALID=val_generator.n//val_generator.batch_size
      print('[LOG] Intialize Neural Network')

      model = Sequential()
      model.add(Conv2D(filters=6,  kernel_size=(3,  3),  activation='relu',
```

```
input_shape=(IMG_SIZE,IMG_SIZE,1)))
      model.add(AveragePooling2D())
      model.add(Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
      model.add(AveragePooling2D())
      model.add(Flatten())
      model.add(Dense(units=120, activation='relu'))
      model.add(Dense(units=84, activation='relu'))
      model.add(Dense(units=1, activation = 'sigmoid'))

      model.compile(loss='binary_crossentropy',            optimizer='adam',
metrics=['accuracy'])
      model.fit_generator(generator=train_generator,
                          steps_per_epoch=STEP_SIZE_TRAIN,
                          validation_data=val_generator,
                          validation_steps=STEP_SIZE_VALID,
                          epochs=20
      )
      save_model(model)
```

Conforme o código acima, o processo abre o arquivo para salvar e começa a executar o treinamento de acordo com as imagens disponibilizadas no Dataset.

Como é feito para executar o que foi treinado? Nesse caso, lembram-se do arquivo "AnomalyDetector.h5"? Ele guardou os pesos da rede treinada, e, a cada entrada de uma nova imagem, a aproximação dos pesos dirá a que classe a imagem pertence.

Abaixo, seguem imagens que explicam como utilizar. Todos os códigos foram implementados pelo desenvolvedor Paulo Vitor Pereira Cotta e estão disponibilizados no GtiHub: https://github.com/paulovpcotta:

```
import os
import cv2
import face_recognition
import numpy as np
from tqdm import tqdm
from collections import defaultdict
from imutils.video import VideoStream
from eye_status import *
def init():
    face_cascPath = 'haarcascade_frontalface_alt.xml'
    #face_cascPath = 'lbpcascade_frontalface.xml'
    open_eye_cascPath = 'haarcascade_eye_tree_eyeglasses.xml'
    left_eye_cascPath = 'haarcascade_lefteye_2splits.xml'
    right_eye_cascPath ='haarcascade_righteye_2splits.xml'
    dataset = 'faces'
    face_detector = cv2.CascadeClassifier(face_cascPath)
    open_eyes_detector = cv2.CascadeClassifier(open_eye_cascPath)
    left_eye_detector = cv2.CascadeClassifier(left_eye_cascPath)
    right_eye_detector = cv2.CascadeClassifier(right_eye_cascPath)
    print("[LOG] Opening webcam ...")
    video_capture = VideoStream(src=0).start()
```

```
    model = load_model()
    print("[LOG] Collecting images ...")
    images = []
    for direc, _, files in tqdm(os.walk(dataset)):
        for file in files:
            if file.endswith("jpg"):
                images.append(os.path.join(direc,file))
   return (model,face_detector, open_eyes_detector, left_eye_detector,right_
eye_detector, video_capture, images)
def process_and_encode(images):
    # initialize the list of known encodings and known names
    known_encodings = []
    known_names = []
    print("[LOG] Encoding faces ...")
    for image_path in tqdm(images):
        # Load image
        image = cv2.imread(image_path)
        # Convert it from BGR to RGB
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
           # detect face in the image and get its location (square boxes
coordinates)
        boxes = face_recognition.face_locations(image, model='hog')
        # Encode the face into a 128-d embeddings vector
        encoding = face_recognition.face_encodings(image, boxes)
        # the person's name is the name of the folder where the image comes
from
        name = image_path.split(os.path.sep)[-2]
        if len(encoding) > 0 :
            known_encodings.append(encoding[0])
            known_names.append(name)
    return {"encodings": known_encodings, "names": known_names}
def isBlinking(history, maxFrames):
    """ @history: A string containing the history of eyes status
         where a '1' means that the eyes were closed and '0' open.
        @maxFrames: The maximal number of successive frames where an eye is
closed """
    for i in range(maxFrames):
        pattern = '1' + '0'*(i+1) + '1'
        if pattern in history:
            return True
    return False
def  detect_and_display(model,  video_capture,  face_detector,  open_eyes_
detector, left_eye_detector, right_eye_detector, data, eyes_detected):
        status = False
        frame = video_capture.read()
        # resize the frame
        frame = cv2.resize(frame, (0, 0), fx=0.6, fy=0.6)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Detect faces
        faces = face_detector.detectMultiScale(
            gray,
```

```
            scaleFactor=1.2,
            minNeighbors=5,
            minSize=(50, 50),
            flags=cv2.CASCADE_SCALE_IMAGE
        )
        # for each detected face
        for (x,y,w,h) in faces:
            # Encode the face into a 128-d embeddings vector
             encoding = face_recognition.face_encodings(rgb, [(y, x+w, y+h,
x)])[0]
            # Compare the vector with all known faces encodings
                matches = face_recognition.compare_faces(data["encodings"],
encoding)
            # For now we don't know the person name
            name = "Unknown"
            # If there is at least one match:
            if True in matches:
                matchedIdxs = [i for (i, b) in enumerate(matches) if b]
                counts = {}
                for i in matchedIdxs:
                    name = data["names"][i]
                    counts[name] = counts.get(name, 0) + 1
                 # determine the recognized face with the largest number of
votes
                name = max(counts, key=counts.get)
            face = frame[y:y+h,x:x+w]
            gray_face = gray[y:y+h,x:x+w]
            eyes = []

            # Eyes detection
            # check first if eyes are open (with glasses taking into account)
            open_eyes_glasses = open_eyes_detector.detectMultiScale(
                gray_face,
                scaleFactor=1.1,
                minNeighbors=5,
                minSize=(30, 30),
                flags = cv2.CASCADE_SCALE_IMAGE
            )
            # if open_eyes_glasses detect eyes then they are open
            if len(open_eyes_glasses) == 2:
                eyes_detected[name]+='1'
                status = True
                for (ex,ey,ew,eh) in open_eyes_glasses:
                    cv2.rectangle(face,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

            # otherwise try detecting eyes using left and right_eye_detector
            # which can detect open and closed eyes
            else:
                # separate the face into left and right sides
                left_face = frame[y:y+h, x+int(w/2):x+w]
                left_face_gray = gray[y:y+h, x+int(w/2):x+w]
                right_face = frame[y:y+h, x:x+int(w/2)]
                right_face_gray = gray[y:y+h, x:x+int(w/2)]
```

```
                # Detect the left eye
                left_eye = left_eye_detector.detectMultiScale(
                    left_face_gray,
                    scaleFactor=1.1,
                    minNeighbors=5,
                    minSize=(30, 30),
                    flags = cv2.CASCADE_SCALE_IMAGE
                )
                # Detect the right eye
                right_eye = right_eye_detector.detectMultiScale(
                    right_face_gray,
                    scaleFactor=1.1,
                    minNeighbors=5,
                    minSize=(30, 30),
                    flags = cv2.CASCADE_SCALE_IMAGE
                )
                eye_status = '1' # we suppose the eyes are open
                # For each eye check wether the eye is closed.
                # If one is closed we conclude the eyes are closed
                for (ex,ey,ew,eh) in right_eye:
                    color = (0,255,0)
                    pred = predict(right_face[ey:ey+eh,ex:ex+ew],model)
                    if pred == 'closed':
                        eye_status='0'
                        color = (0,0,255)
                    cv2.rectangle(right_face,(ex,ey),(ex+ew,ey+eh),color,2)
                for (ex,ey,ew,eh) in left_eye:
                    color = (0,255,0)
                    pred = predict(left_face[ey:ey+eh,ex:ex+ew],model)
                    if pred == 'closed':
                        eye_status='0'
                        color = (0,0,255)
                    cv2.rectangle(left_face,(ex,ey),(ex+ew,ey+eh),color,2)
                eyes_detected[name] += eye_status
                status = False
            # Each time, we check if the person has blinked
            # If yes, we display its name
            if isBlinking(eyes_detected[name],3):
                cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
                # Display name
                y = y - 15 if y - 15 > 15 else y + 15
             cv2.putText(frame, name, (x, y), cv2.FONT_HERSHEY_SIMPLEX,0.75,
(0, 255, 0), 2)
        return frame, status
if __name__ == "__main__":
     (model, face_detector, open_eyes_detector,left_eye_detector,right_eye_
detector, video_capture, images) = init()
    data = process_and_encode(images)
    eyes_detected = defaultdict(str)
    while True:
            frame, status = detect_and_display(model, video_capture, face_
detector,  open_eyes_detector,left_eye_detector,right_eye_detector,  data,
eyes_detected)
```

```
        cv2.imshow("Face Liveness Detector", frame)
        print(status)
        if(status):
            break
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cv2.destroyAllWindows()
    video_capture.stop()
```

Nesse ponto, foi usado o modelo treinado "AnomalyDetector.h5". Quando a rede fez o *load* dos pesos, a função frame_gray pegou e recebeu o modelo carregado para ser usado. Nesse caso, quando identifica os conjuntos de cada item diferente, ele calcula a distância e recupera a classe a que a imagem pertence.

Nesse algoritmo, foi utilizada uma técnica chamada *threshold* ou, em português, limite. Esse limite é o mínimo para o item que queremos identificar. Se estiver acima do limite, ele identifica como a classe de anomalia, que é o caso do algoritmo.

Nesse caso, após treinado, executado e criada a API, o algoritmo está preparado para ir para produção.

Documentação do Python 3.7

O Python tem uma documentação muito boa (https://docs.python.org/3.7/), porém atualmente não está disponível para o idioma Português Brasil.

Nesse processo, deve-se lembrar que a linguagem Python é uma linguagem simples e muito robusta de aprender e fazer *software*.

A linguagem Python tem como perfil mostrar suas facilidades. Dessa forma, a comunidade brasileira de Python traduziu a documentação, conforme está apresentado abaixo:

- *site* da documentação do Python Brasil: https://wiki.python.org.br/DocumentacaoPython;
- complementação da documentação em Português Brasil: https://wiki.python.org.br/PythonDoc.

CAPÍTULO 2

CÓDIGO FONTE PARA EXEMPLIFICAR

Código fonte para apresentar IA

De acordo com GitHub (https://github.com/paulovpcotta), temos exemplos de alguns algoritmos que podem facilitar a construção de algoritmos de *Machine Learning* e *Deep Learning*.

GAN com MNIST

Conforme explicado no capítulo de *Deep Learning* GAN Redes Neurais Generativas e Adversárias, segue um exemplo de código fonte em Python, versão final apresentada no link https://github.com/paulovpcotta/face_generation, e a base de dados http://yann.lecun.com/exdb/mnist/ e http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html:

```
data_dir = './data'
import helper
helper.download_extract('mnist', data_dir)
helper.download_extract('celeba', data_dir)
show_n_images = 25
%matplotlib inline
import os
from glob import glob
from matplotlib import pyplot
mnist_images = helper.get_batch(glob(os.path.join(data_dir, 'mnist/*.jpg'))
[show_n_images:2*show_n_images], 28, 28, 'L')
pyplot.imshow(helper.images_square_grid(mnist_images, 'L'), cmap='gray')
mnist_images = helper.get_batch(glob(os.path.join(data_dir, 'mnist/*.jpg'))
[show_n_images:2*show_n_images], 28, 28, 'L')
show_n_images = 25
mnist_images = helper.get_batch(glob(os.path.join(data_dir, 'img_align_
celeba/*.jpg'))[:show_n_images], 28, 28, 'RGB')
pyplot.imshow(helper.images_square_grid(mnist_images, 'RGB'))
from distutils.version import LooseVersion
import warnings
import tensorflow as tf
# Check TensorFlow Version
assert LooseVersion(tf.__version__) >= LooseVersion('1.0'), 'Please use
TensorFlow version 1.0 or newer. You are using {}'.format(tf.__version__)
print('TensorFlow Version: {}'.format(tf.__version__))
# Check for a GPU
if not tf.test.gpu_device_name():
     warnings.warn('No GPU found. Please use a GPU to train your neural
network.')
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
import problem_unittests as tests
```

```
def model_inputs(image_width, image_height, image_channels, z_dim):
    """
    Create the model inputs
    :param image_width: The input image width
    :param image_height: The input image height
    :param image_channels: The number of image channels
    :param z_dim: The dimension of Z
    :return: Tuple of (tensor of real input images, tensor of z data, learning
rate)
    """
    # TODO: Implement Function
      inputs_real=tf.placeholder(tf.float32,(None,image_width,  image_height,
image_channels),name='inputs_real')
    inputs_z=tf.placeholder(tf.float32,(None,z_dim),name='inputs_z')
    learningrate=tf.placeholder(tf.float32,name='learningrate')
    return inputs_real, inputs_z, learningrate
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_inputs(model_inputs)
def discriminator(images, reuse=False,alpha=0.2):
    """
    Create the discriminator network
    :param image: Tensor of input image(s)
    :param reuse: Boolean if the weights should be reused
    :return: Tuple of (tensor output of the discriminator, tensor logits of
the discriminator)
    """
    # TODO: Implement Function
    with tf.variable_scope('discriminator', reuse=reuse):
        init=tf.random_normal_initializer(mean=0,stddev=0.02)
        # Input layer is 28x28x3
         h1=tf.layers.conv2d(images,  64,5,strides=2,  padding='SAME',kernel_
initializer=init)
        relu1=tf.maximum(alpha*h1,h1)
        # 14x14x64

h2=tf.layers.conv2d(relu1,128,5,strides=2,padding="SAME",kernel_
initializer=init)
        h2=tf.layers.batch_normalization(h2,training=True)
        h2=tf.maximum(alpha*h2,h2)
        # 7x7x128
h3=tf.layers.conv2d(h2,256,5,strides=1,padding="VALID",kernel_
initializer=init)
        h3=tf.layers.batch_normalization(h3,training=True)
        relu3=tf.maximum(alpha*h3,h3)
        # 3x3x256
        # 1x1x512
        flat = tf.reshape(h2, (-1, 7*7*128))

        logit=tf.layers.dense(flat,1)
        out=tf.sigmoid(logit)
    return out, logit
```

```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_discriminator(discriminator, tf)
def generator(z, out_channel_dim, is_train=True,reuse=False,alpha=0.01):
    """
    Create the generator network
    :param z: Input z
    :param out_channel_dim: The number of channels in the output image
    :param is_train: Boolean if generator is being used for training
    :return: The tensor output of the generator
    """
    # TODO: Implement Function
    reuse = not is_train

    with tf.variable_scope('generator',reuse=reuse):
#     with tf.variable_scope('generator'):

#     with tf.variable_scope('generator', reuse=reuse):
        init= tf.contrib.layers.xavier_initializer()
        h1=tf.layers.dense(z,7*7*512,kernel_initializer=init)
        h1=tf.reshape(h1,(-1,7,7,512))
        h1=tf.layers.batch_normalization(h1,training=is_train)
        h1=tf.maximum(alpha*h1,h1)
        # 3x3x512
                h2  =  tf.layers.conv2d_transpose(h1,  256,  3,  strides=2,
padding='SAME',kernel_initializer=init)
        h2=tf.layers.batch_normalization(h2,training=is_train)
        h2=tf.maximum(alpha*h2,h2)

#         print(h2,t1)

        # 7x7x256
                h3  =  tf.layers.conv2d_transpose(h1,  128,  3,  strides=2,
padding='SAME',kernel_initializer=init)
        h3=tf.layers.batch_normalization(h3,training=is_train)
        h3=tf.maximum(alpha*h3,h3)
        # 14x14x128
       logits = tf.layers.conv2d_transpose(h3, out_channel_dim, 3, strides=2,
padding='SAME',kernel_initializer=init)
        # 28x28x3
        out=tf.tanh(logits)
        # 28x28x3
    return out
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_generator(generator, tf)
def model_loss(input_real, input_z, out_channel_dim,alpha=0.01):
    """
    Get the loss for the discriminator and generator
    :param input_real: Images from the real dataset
```

```
    :param input_z: Z input
    :param out_channel_dim: The number of channels in the output image
    :return: A tuple of (discriminator loss, generator loss)
    """
    # TODO: Implement Function
              g_model=generator(input_z,out_channel_dim=out_channel_dim,is_
train=True,alpha = 0.01)
    d_model_real,d_logits_real=discriminator(input_real,alpha = 0.01)
   d_model_fake,d_logits_fake=discriminator(g_model,reuse=True,alpha = 0.01)

    d_loss_real=tf.reduce_mean(
     tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_real,labels=tf.
ones_like(d_logits_real)* 0.9))
    d_loss_fake=tf.reduce_mean(
tf.nn.sigmoid_cross_entropy_with_logits(logits=d_logits_fake,labels=tf.
zeros_like(d_logits_fake)))
     g_loss=tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=d_
logits_fake,labels=tf.ones_like(d_logits_fake)*0.9))
    d_loss=d_loss_real+d_loss_fake
    return d_loss, g_loss
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_loss(model_loss)
def model_opt(d_loss, g_loss, learning_rate, beta1):
    """
    Get optimization operations
    :param d_loss: Discriminator loss Tensor
    :param g_loss: Generator loss Tensor
    :param learning_rate: Learning Rate Placeholder
      :param beta1: The exponential decay rate for the 1st moment in the
optimizer
    :return: A tuple of (discriminator training operation, generator training
operation)
    """
    # TODO: Implement Function
    t_var=tf.trainable_variables()
    d_var=[var for var in t_var if var.name.startswith('discriminator')]
    g_var=[var for var in t_var if var.name.startswith('generator')]

   with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):
       d_train_op=tf.train.AdamOptimizer(beta1=beta1,learning_rate=learning_
rate).minimize(d_loss,var_list=d_var)

       g_train_op=tf.train.AdamOptimizer(beta1=beta1,learning_rate=learning_
rate).minimize(g_loss,var_list=g_var)
  return d_train_op, g_train_op
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_opt(model_opt, tf)
"""
DON'T MODIFY ANYTHING IN THIS CELL
```

```
"""
import numpy as np
def show_generator_output(sess, n_images, input_z, out_channel_dim, image_
mode):
    """
    Show example output for the generator
    :param sess: TensorFlow session
    :param n_images: Number of Images to display
    :param input_z: Input Z Tensor
    :param out_channel_dim: The number of channels in the output image
    :param image_mode: The mode to use for images ("RGB" or "L")
    """
    cmap = None if image_mode == 'RGB' else 'gray'
    z_dim = input_z.get_shape().as_list()[-1]
    example_z = np.random.uniform(-1, 1, size=[n_images, z_dim])
    samples = sess.run(
        generator(input_z, out_channel_dim, False),
        feed_dict={input_z: example_z})
    images_grid = helper.images_square_grid(samples, image_mode)
    pyplot.imshow(images_grid, cmap=cmap)
    pyplot.show()
def train(epoch_count, batch_size, z_dim, learning_rate, beta1, get_batches,
data_shape, data_image_mode,alpha,writer):
    """
    Train the GAN
    :param epoch_count: Number of epochs
    :param batch_size: Batch Size
    :param z_dim: Z dimension
    :param learning_rate: Learning Rate
     :param beta1: The exponential decay rate for the 1st moment in the
optimizer
    :param get_batches: Function to get batches
    :param data_shape: Shape of the data
    :param data_image_mode: The image mode to use for images ("RGB" or "L")
    """
    # TODO: Build Model
    image_width, image_height, image_channels=data_shape[1:]

   input_real, input_z,learningrate=model_inputs(image_width, image_height,
image_channels, z_dim)

  d_loss, g_loss=model_loss(input_real, input_z, image_channels,alpha=alpha)
    d_train_op, g_train_op=model_opt(d_loss, g_loss, learningrate, beta1)

    saver = tf.train.Saver()

    steps = 0
    sc=1
    with tf.Session() as sess:
        tf.summary.scalar('d_loss',d_loss)
        tf.summary.scalar('g_loss',g_loss)
        merge_summary=tf.summary.merge_all()
        sess.run(tf.global_variables_initializer())
```

```
        writer.add_graph(sess.graph)
        for epoch_i in range(epoch_count):

            for batch_images in get_batches(batch_size):
                # TODO: Train Model
                steps+=1

              batch_z = np.random.uniform(-1, 1, size=(1*batch_size, z_dim))
                batch_images=batch_images*2
                            feed_dict={input_real:  batch_images,  input_z:
batch_z,learningrate:learning_rate}
              train_loss_d,_ = sess.run([d_loss,d_train_op], feed_dict=feed_
dict)
              train_loss_g,_ = sess.run([g_loss,g_train_op], feed_dict=feed_
dict)
              train_loss_g,_ = sess.run([g_loss,g_train_op], feed_dict=feed_
dict)

                if train_loss_d>train_loss_g:
                    sess.run(d_train_op, feed_dict=feed_dict)
                else:
                    sess.run(g_train_op, feed_dict=feed_dict)
                if steps % 100 == 0:
                   show_generator_output(sess, 16, input_z, image_channels,
data_image_mode)
                if steps % 20 == 0:

                    train_loss_d = sess.run(d_loss, feed_dict)
                    train_loss_g = sess.run(g_loss, feed_dict)
                    s=sess.run(merge_summary,feed_dict)
                    writer.add_summary(s,steps)

                   print("Step : {} Epoch {}/{}...".format(steps,epoch_i+1,
epoch_count),
                  "Discriminator Loss: {:.4f}...".format(train_loss_d),
                  "Generator Loss: {:.4f}".format(train_loss_g))
                    losses.append((train_loss_d, train_loss_g))
!rm -r /tmp/gan
batch_size = 64
z_dim = 64
learning_rate = 0.0003
beta1 = 0.4
alpha=0.01
losses = []
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
epochs = 2
writer=tf.summary.FileWriter('/tmp/gan/mnist')
mnist_dataset = helper.Dataset('mnist', glob(os.path.join(data_dir, 'mnist/*.
jpg')))
with tf.Graph().as_default():
```

```
    train(epochs, batch_size, z_dim, learning_rate, beta1, mnist_dataset.
get_batches,
          mnist_dataset.shape, mnist_dataset.image_mode,alpha,writer)
fig, ax = pyplot.subplots()
losses = np.array(losses)
pyplot.plot(losses.T[0], label='Discriminator')
pyplot.plot(losses.T[1], label='Generator')
pyplot.title("Training Losses")
pyplot.legend()
batch_size = 64
z_dim = 128
learning_rate = 0.0002
beta1 = 0.5
alpha=0.01

losses = []
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
epochs = 1
writer=tf.summary.FileWriter('/tmp/gan/face')
celeba_dataset = helper.Dataset('celeba', glob(os.path.join(data_dir, 'img_
align_celeba/*.jpg')))
with tf.Graph().as_default():
    train(epochs, batch_size, z_dim, learning_rate, beta1, celeba_dataset.
get_batches,
          celeba_dataset.shape, celeba_dataset.image_mode,alpha,writer)
fig, ax = pyplot.subplots()
losses = np.array(losses)
pyplot.plot(losses.T[0], label='Discriminator')
pyplot.plot(losses.T[1], label='Generator')
pyplot.title("Training Losses")
pyplot.legend()
```

Regressão Linear

Esse código foi feito no momento em que estava executando um curso da Udacity. Ele demonstra o processo de regressão linear e como funciona (https://github.com/paulovpcotta/boston_housing_udacity). A base de dados é baseada na UCI (https://archive.ics.uci.edu/ml/datasets/Housing). Abaixo segue o código fonte:

```
import sklearn
print("A versão do scikit-learn é ", sklearn.__version__)
if sklearn.__version__ >= '0.18':
    print("Tudo certo!")
else:
    print("Você precisa fazer upgrade do scikit-learn ou ficar atento com as
diferenças das versões")
    print("Pode ser feito executando:\n")
    print("pip install scikit-learn==0.18.1")
```

```
# Importar as bibliotecas necessárias para este projeto
import numpy as np
import pandas as pd
import visuals as vs # Supplementary code
from sklearn.model_selection import ShuffleSplit
# Formatação mais bonita para os notebooks
%matplotlib inline
# Executar o conjunto de dados de imóveis de Boston
data = pd.read_csv('housing.csv')
prices = data['MEDV']
features = data.drop('MEDV', axis = 1)
data.info()
# Êxito
print("O conjunto de dados de imóveis de Boston tem {} pontos com {} variáveis
em cada.".format(*data.shape))
# TODO: Preço mínimo dos dados
minimum_price = np.amin(prices)
# TODO: Preço máximo dos dados
maximum_price = np.amax(prices)
# TODO: Preço médio dos dados
mean_price = np.mean(prices)
# TODO: Preço mediano dos dados
median_price = np.median(prices)
# TODO: Desvio padrão do preço dos dados
std_price = np.std(prices)
# Mostrar as estatísticas calculadas
print("Estatísticas para os dados dos imóveis de Boston:\n")
print("Preço mínimo: ${:,.2f}".format(minimum_price))
print("Preço máximo: ${:,.2f}".format(maximum_price))
print("Preço médio: ${:,.2f}".format(mean_price))
print("Preço mediano: ${:,.2f}".format(median_price))
print("Desvio padrão dos preços: ${:,.2f}".format(std_price))
data.describe()
# TODO: Importar 'r2_score'
from sklearn.metrics import r2_score
def performance_metric(y_true, y_predict):
    """ Calcular e retornar a pontuação de desempenho entre
        valores reais e estimados baseado na métrica escolhida. """

    # TODO: Calcular a pontuação de desempenho entre 'y_true' e 'y_predict'
    score = r2_score(y_true, y_predict)

    # Devolver a pontuação
    return score
# Calcular o desempenho deste modelo
score = performance_metric([3, -0.5, 2, 7, 4.2], [2.5, 0.0, 2.1, 7.8, 5.3])
print("O coeficiente de determinação, R^2, do modelo é {:.3f}.".format(score))
# TODO: Importar 'train_test_split'
from sklearn.model_selection import train_test_split
# TODO: Misturar e separar os dados em conjuntos de treinamento e teste
X = np.array(data[['RM', 'LSTAT', 'PTRATIO']])
y = np.array(data['MEDV'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

```
random_state=42)
# Êxito
print("Separação entre treino e teste feita com êxito.")
# Criar curvas de aprendizagem para tamanhos de conjunto de treinamento
variável e profundidades máximas
vs.ModelLearning(features, prices)
vs.ModelComplexity(X_train, y_train)
# TODO: Importar 'make_scorer', 'DecisionTreeRegressor' e 'GridSearchCV'
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import make_scorer
def fit_model(X, y):
    """ Desempenhar busca em matriz sobre o parâmetro the 'max_depth' para uma
        árvore de decisão de regressão treinada nos dados de entrada [X, y].
"""

    # Gerar conjuntos de validação-cruzada para o treinamento de dados
    cv_sets = ShuffleSplit(n_splits=10, test_size = 0.20, random_state=0)
    # TODO: Gerar uma árvore de decisão de regressão de objeto
    regressor = DecisionTreeRegressor()
    # TODO: Gerar um dicionário para o parâmetro 'max_depth' com um alcance
de 1 a 10
    params = {'max_depth': list(range(1, 11))}
      # TODO: Transformar 'performance_metric' em uma função de pontuação
utilizando 'make_scorer'
    scoring_fnc = make_scorer(performance_metric)
    # TODO: Gerar o objeto de busca em matriz
    grid = GridSearchCV(cv=cv_sets, estimator=regressor, param_grid=params,
scoring=scoring_fnc)
    # Ajustar o objeto de busca em matriz com os dados para calcular o modelo
ótimo
    grid = grid.fit(X, y)
    # Devolver o modelo ótimo depois de realizar o ajuste dos dados
    return grid.best_estimator_
# Ajustar os dados de treinamento para o modelo utilizando busca em matriz
reg = fit_model(X_train, y_train)
# Produzir valores para 'max_depth'
print("O parâmetro 'max_depth' é {} para o modelo ótimo.".format(reg.get_
params()['max_depth']))
# Gerar uma matriz para os dados do cliente
client_data = [[5, 17, 15], # Cliente 1
               [4, 32, 22], # Cliente 2
               [8, 3, 12]]  # Cliente 3
# Mostrar estimativas
for i, price in enumerate(reg.predict(client_data)):
    print("Preço estimado para a casa do cliente {}: ${:,.2f}".format(i+1,
price))
vs.PredictTrials(features, prices, fit_model, client_data)
```

Processo de classificação com *Machine Learning*

A ideia do código fonte é apresentar como um processo de classificar funciona. O GitHub original é https://github.com/paulovpcotta/diabete-igti-deep-learning, e a base de dados é da Udacity.com. O modelo foi feito dentro do *framework* Keras.

```
# imports
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import numpy
import pandas as pd
# Importação do DataSet
dataset = pd.read_csv('diabetes.csv')
dataset.head()
#X = dataset[:,0:8]
#### Pega apenas as primeiras 8 colunas conforme solicitado
X = dataset.drop('Outcome',axis=1)
print(len(X))
X.head()
#Y = dataset[:,8]
Y = dataset['Outcome']
print(len(Y))
Y.head()
# Preparando o treino dos dados
################################################
### Processo de criação das variáveis para treino,
### levando em conta que o test_size é o learning
### rate para esse aprendizado
################################################
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
# Processo de preparação preditiva
model = Sequential()
model.add(Dense(12, input_dim=8, init='uniform',activation='relu'))
model.add(Dense(8, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))
model.compile(loss='binary_crossentropy'          ,          optimizer='adam',
metrics=['accuracy'])
# Irá pegar 10 itens do DataSet e depois a cada iteração fará 150 épocas no
processo de treino
model.fit(X_train, y_train, nb_epoch=150, batch_size=10)
# Avaliação da acurácia e do Log_loss
scores = model.evaluate(X_test, y_test)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

Reinforcement Learning

Temos um processo de aprendizado por reforço em que a máquina aprende errando até o momento de acertar. Nesse caso, vamos obter um código baseado em um Quadcopter

100% autônomo que voa a partir de alguns erros que ele comete, lembrando que é uma simulação a partir do conjunto de dados. O link do GitHub é https://github.com/paulovpcotta/quadcopter, e o conjunto de dados é Udacity.com.

```
import random
class Basic_Agent():
    def __init__(self, task):
        self.task = task

    def act(self):
        new_thrust = random.gauss(450., 25.)
        return [new_thrust + random.gauss(0., 1.) for x in range(4)]
%load_ext autoreload
%autoreload 2
import csv
import numpy as np
from task import Task
# Modify the values below to give the quadcopter a different starting position.
runtime = 5.                                     # time limit of the episode
init_pose = np.array([0., 0., 10., 0., 0., 0.])  # initial pose
init_velocities = np.array([0., 0., 0.])         # initial velocities
init_angle_velocities = np.array([0., 0., 0.])   # initial angle velocities
file_output = 'data.txt'                         # file name for saved results
# Setup
task = Task(init_pose, init_velocities, init_angle_velocities, runtime)
agent = Basic_Agent(task)
done = False
labels = ['time', 'x', 'y', 'z', 'phi', 'theta', 'psi', 'x_velocity',
          'y_velocity', 'z_velocity', 'phi_velocity', 'theta_velocity',
            'psi_velocity', 'rotor_speed1', 'rotor_speed2', 'rotor_speed3',
'rotor_speed4']
results = {x : [] for x in labels}
# Run the simulation, and save the results.
with open(file_output, 'w') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(labels)
    while True:
        rotor_speeds = agent.act()
        _, _, done = task.step(rotor_speeds)
        to_write = [task.sim.time] + list(task.sim.pose) + list(task.sim.v)
+ list(task.sim.angular_v) + list(rotor_speeds)
        for ii in range(len(labels)):
            results[labels[ii]].append(to_write[ii])
        writer.writerow(to_write)
        if done:
            break
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(results['time'], results['x'], label='x')
plt.plot(results['time'], results['y'], label='y')
plt.plot(results['time'], results['z'], label='z')
plt.legend()
_ = plt.ylim()
```

```
plt.plot(results['time'], results['x_velocity'], label='x_hat')
plt.plot(results['time'], results['y_velocity'], label='y_hat')
plt.plot(results['time'], results['z_velocity'], label='z_hat')
plt.legend()
_ = plt.ylim()
plt.plot(results['time'], results['phi'], label='phi')
plt.plot(results['time'], results['theta'], label='theta')
plt.plot(results['time'], results['psi'], label='psi')
plt.legend()
_ = plt.ylim()
plt.plot(results['time'], results['phi_velocity'], label='phi_velocity')
plt.plot(results['time'], results['theta_velocity'], label='theta_velocity')
plt.plot(results['time'], results['psi_velocity'], label='psi_velocity')
plt.legend()
_ = plt.ylim()
plt.plot(results['time'], results['rotor_speed1'], label='Rotor 1 revolutions
/ second')
plt.plot(results['time'], results['rotor_speed2'], label='Rotor 2 revolutions
/ second')
plt.plot(results['time'], results['rotor_speed3'], label='Rotor 3 revolutions
/ second')
plt.plot(results['time'], results['rotor_speed4'], label='Rotor 4 revolutions
/ second')
plt.legend()
_ = plt.ylim()
# the pose, velocity, and angular velocity of the quadcopter at the end of
the episode
print(task.sim.pose)
print(task.sim.v)
print(task.sim.angular_v)
import sys
import pandas as pd
from agents.policy_search import PolicySearch_Agent
from task import Task
num_episodes = 4000
target_pos = np.array([0., 0., 10.])
task = Task(target_pos=target_pos)
agent = PolicySearch_Agent(task)
for i_episode in range(1, num_episodes+1):
    state = agent.reset_episode() # start a new episode
    while True:
        action = agent.act(state)
        next_state, reward, done = task.step(action)
        agent.step(reward, done)
        state = next_state
        if done:
               print("\rEpisode = {:4d}, score = {:7.3f} (best = {:7.3f}),
noise_scale = {}".format(
                i_episode, agent.score, agent.best_score, agent.noise_scale),
end="")  # [debug]
            break
    sys.stdout.flush()
import sys
```

```
import pandas as pd
import csv
import numpy as np
from agents.agent import DDPG
from task import Task
# HOVER TASK
# Modify the values below to give the quadcopter a different starting position.
runtime = 5.                                     # time limit of the episode
init_pose = np.array([0., 0., 10., 0., 0., 0.])  # initial pose
init_velocities = np.array([0., 0., 0.])         # initial velocities
init_angle_velocities = np.array([0., 0., 0.])   # initial angle velocities
file_output = 'rewards.txt'                      # file name for saved results
num_episodes = 1500
target_pos = np.array([0., 0., 10.])
task = Task(target_pos=target_pos)
agent = DDPG(task)
labels = ['e2', 't2']
results = {x : [] for x in labels}
with open(file_output, 'w') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(labels)
    best_total_reward = 0
    for i_episode in range(1, num_episodes+1):
        state = agent.reset_episode() # start a new episode
        total_reward = 0
        while True:
            action = agent.act(state)
            next_state, reward, done = task.step(action)
            total_reward += reward
            if total_reward > best_total_reward:
                best_total_reward = total_reward
            agent.step(action, reward, next_state, done)
            state = next_state
            if done:
                to_write = [i_episode] + [total_reward]
                for ii in range(len(labels)):
                    results[labels[ii]].append(to_write[ii])
                writer.writerow(to_write)
                  print("\rEpisode = {:4d}, total_reward = {:7.3f} (best =
{:7.3f})".format(
                    i_episode, total_reward, best_total_reward), end="")
                break
        sys.stdout.flush()
import sys
import pandas as pd
import csv
import numpy as np
from agents.agent import DDPG
from task import Task
# HOVER TASK
# Modify the values below to give the quadcopter a different starting position.
runtime = 5.                                     # time limit of the episode
init_pose = np.array([0., 0., 10., 0., 0., 0.])  # initial pose
```

```
init_velocities = np.array([0., 0., 0.])          # initial velocities
init_angle_velocities = np.array([0., 0., 0.])    # initial angle velocities
file_output = 'rewards.txt'                       # file name for saved results
num_episodes = 2000
target_pos = np.array([0., 0., 10.])
task = Task(target_pos=target_pos)
agent = DDPG(task)
labels = ['e1', 't1']
results = {x : [] for x in labels}
with open(file_output, 'w') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(labels)
    best_total_reward = 0
    for i_episode in range(1, num_episodes+1):
        state = agent.reset_episode() # start a new episode
        total_reward = 0
        while True:
            action = agent.act(state)
            next_state, reward, done = task.step(action)
            total_reward += reward
            if total_reward > best_total_reward:
                best_total_reward = total_reward
            agent.step(action, reward, next_state, done)
            state = next_state
            if done:
                to_write = [i_episode] + [total_reward]
                for ii in range(len(labels)):
                    results[labels[ii]].append(to_write[ii])
                writer.writerow(to_write)
                  print("\rEpisode = {:4d}, total_reward = {:7.3f} (best =
{:7.3f})".format(
                    i_episode, total_reward, best_total_reward), end="")
                break
        sys.stdout.flush()
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(results['e1'], results['t1'])
#plt.legend()
_ = plt.ylim()
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(results['e2'], results['t2'])
#plt.legend()
_ = plt.ylim()
```

NLP exemplo básico sem análise de sentimento e emoção

O processo de identificação textual e transcrever para outro idioma, ou entender a frase é conhecido como análise sentimental e emocional. Nesse caso, não com o que lidaremos neste momento, mas será apresentado no GitHub https://github.com/

paulovpcotta/language_translation o processo de tradução usando *Deep Learning* com o conceito de NLP *Natural Leanguage Proccess,* e o conjunto de dados é da Udacity. com, tradução do inglês para o francês. Segue o código abaixo:

```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import helper
import problem_unittests as tests
source_path = 'data/small_vocab_en'
target_path = 'data/small_vocab_fr'
source_text = helper.load_data(source_path)
target_text = helper.load_data(target_path)
view_sentence_range = (0, 10)
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import numpy as np
print('Dataset Stats')
print('Roughly the number of unique words: {}'.format(len({word: None for
word in source_text.split()})))
sentences = source_text.split('\n')
word_counts = [len(sentence.split()) for sentence in sentences]
print('Number of sentences: {}'.format(len(sentences)))
print('Average number of words in a sentence: {}'.format(np.average(word_
counts)))
print()
print('English sentences {} to {}:'.format(*view_sentence_range))
print('\n'.join(source_text.split('\n')[view_sentence_range[0]:view_
sentence_range[1]]))
print()
print('French sentences {} to {}:'.format(*view_sentence_range))
print('\n'.join(target_text.split('\n')[view_sentence_range[0]:view_
sentence_range[1]]))
def text_to_ids(source_text, target_text, source_vocab_to_int, target_vocab_
to_int):
    """
    Convert source and target text to proper word ids
    :param source_text: String that contains all the source text.
    :param target_text: String that contains all the target text.
    :param source_vocab_to_int: Dictionary to go from the source words to an
id
    :param target_vocab_to_int: Dictionary to go from the target words to an
id
    :return: A tuple of lists (source_id_text, target_id_text)
    """
    # TODO: Implement Function
    source_text_ids = list()
    target_text_ids = list()

    source_text_sentences = source_text.split('\n')
    target_text_sentences = target_text.split('\n')
```

```
    for sentence in source_text_sentences:
        sentence_ids = list()
        for word in sentence.split(' '):
            try:
                sentence_ids.append(source_vocab_to_int[word])
            except:
                sentence_ids.append(source_vocab_to_int['<UNK>'])
        source_text_ids.append(sentence_ids)
    for sentence in target_text_sentences:
        sentence = sentence + ' <EOS>'
        sentence_ids = list()
        for word in sentence.split(' '):
            try:
                sentence_ids.append(target_vocab_to_int[word])
            except:
                sentence_ids.append(source_vocab_to_int['<UNK>'])
        target_text_ids.append(sentence_ids)

    return source_text_ids, target_text_ids
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_text_to_ids(text_to_ids)
helper.preprocess_and_save_data(source_path, target_path, text_to_ids)
import numpy as np
import helper
(source_int_text, target_int_text), (source_vocab_to_int, target_vocab_to_
int), _ = helper.load_preprocess()
from distutils.version import LooseVersion
import warnings
import tensorflow as tf
# Check TensorFlow Version
assert     LooseVersion(tf.__version__)     in     [LooseVersion('1.0.0'),
LooseVersion('1.0.1')], 'This project requires TensorFlow version 1.0  You
are using {}'.format(tf.__version__)
print('TensorFlow Version: {}'.format(tf.__version__))
# Check for a GPU
if not tf.test.gpu_device_name():
     warnings.warn('No GPU found. Please use a GPU to train your neural
network.')
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
def model_inputs():
    """
    Create TF Placeholders for input, targets, and learning rate.
    :return: Tuple (input, targets, learning rate, keep probability)
    """
    # TODO: Implement Function
    Input = tf.placeholder(tf.int32, [None, None], name='input')
    Targets = tf.placeholder(tf.int32, [None, None], name='targets')
    LearningRate = tf.placeholder(tf.float32, None, name='learning_rate')
    KeepProbability = tf.placeholder(tf.float32, None, name='keep_prob')
```

```
    return Input, Targets, LearningRate, KeepProbability
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_model_inputs(model_inputs)
def process_decoding_input(target_data, target_vocab_to_int, batch_size):
    """
    Preprocess target data for dencoding
    :param target_data: Target Placehoder
    :param target_vocab_to_int: Dictionary to go from the target words to an
id
    :param batch_size: Batch Size
    :return: Preprocessed target data
    """
    # TODO: Implement Function
    target_data_slice = tf.strided_slice(
        target_data, [0, 0], [batch_size, -1], [1, 1])
    target_data = tf.concat([tf.fill(
       [batch_size, 1], target_vocab_to_int['<GO>']), target_data_slice], 1)
    return target_data
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_process_decoding_input(process_decoding_input)
def encoding_layer(rnn_inputs, rnn_size, num_layers, keep_prob):
    """
    Create encoding layer
    :param rnn_inputs: Inputs for the RNN
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param keep_prob: Dropout keep probability
    :return: RNN state
    """
    # TODO: Implement Function
    enc_cell = tf.contrib.rnn.MultiRNNCell(
        [tf.contrib.rnn.BasicLSTMCell(rnn_size)] * num_layers)
    _, enc_state = tf.nn.dynamic_rnn(
        enc_cell, tf.nn.dropout(rnn_inputs, keep_prob), dtype=tf.float32)
    return enc_state
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_encoding_layer(encoding_layer)
def decoding_layer_train(encoder_state, dec_cell, dec_embed_input, sequence_
length, decoding_scope,
                         output_fn, keep_prob):
    """
    Create a decoding layer for training
    :param encoder_state: Encoder State
    :param dec_cell: Decoder RNN Cell
    :param dec_embed_input: Decoder embedded input
    :param sequence_length: Sequence Length
    :param decoding_scope: TenorFlow Variable Scope for decoding
```

```
    :param output_fn: Function to apply the output layer
    :param keep_prob: Dropout keep probability
    :return: Train Logits
    """
    # TODO: Implement Function
     train_decoder_fn = tf.contrib.seq2seq.simple_decoder_fn_train(encoder_
state)
    train_pred, _, _ = tf.contrib.seq2seq.dynamic_rnn_decoder(
        dec_cell, train_decoder_fn, dec_embed_input,
        sequence_length, scope=decoding_scope)
    train_logits = output_fn(tf.nn.dropout(train_pred, keep_prob))

    return train_logits

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer_train(decoding_layer_train)
def decoding_layer_infer(encoder_state, dec_cell, dec_embeddings, start_of_
sequence_id, end_of_sequence_id,
                         maximum_length, vocab_size, decoding_scope, output_
fn, keep_prob):
    """
    Create a decoding layer for inference
    :param encoder_state: Encoder state
    :param dec_cell: Decoder RNN Cell
    :param dec_embeddings: Decoder embeddings
    :param start_of_sequence_id: GO ID
    :param end_of_sequence_id: EOS Id
    :param maximum_length: The maximum allowed time steps to decode
    :param vocab_size: Size of vocabulary
    :param decoding_scope: TensorFlow Variable Scope for decoding
    :param output_fn: Function to apply the output layer
    :param keep_prob: Dropout keep probability
    :return: Inference Logits
    """
    # TODO: Implement Function
    infer_decoder_fn = tf.contrib.seq2seq.simple_decoder_fn_inference(
        output_fn, encoder_state, dec_embeddings, start_of_sequence_id, end_
of_sequence_id,
        maximum_length - 1, vocab_size)
    infer_pred, _, _ = tf.contrib.seq2seq.dynamic_rnn_decoder(
        dec_cell, infer_decoder_fn, scope=decoding_scope)
    inference_logits = tf.nn.dropout(infer_pred, keep_prob)

    return inference_logits

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer_infer(decoding_layer_infer)
def decoding_layer(dec_embed_input, dec_embeddings, encoder_state, vocab_
size, sequence_length, rnn_size,
```

```
                   num_layers, target_vocab_to_int, keep_prob):
    """
    Create decoding layer
    :param dec_embed_input: Decoder embedded input
    :param dec_embeddings: Decoder embeddings
    :param encoder_state: The encoded state
    :param vocab_size: Size of vocabulary
    :param sequence_length: Sequence Length
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param target_vocab_to_int: Dictionary to go from the target words to an
id
    :param keep_prob: Dropout keep probability
    :return: Tuple of (Training Logits, Inference Logits)
    """
    # TODO: Implement Function
    dec_cell = tf.contrib.rnn.MultiRNNCell(
        [tf.contrib.rnn.BasicLSTMCell(rnn_size)] * num_layers)

    with tf.variable_scope("decoding") as decoding_scope:
        output_fn = lambda x: tf.contrib.layers.fully_connected(
            x, vocab_size, None, scope=decoding_scope)
        train_logits = decoding_layer_train(
            encoder_state, dec_cell, dec_embed_input, sequence_length,
            decoding_scope, output_fn, keep_prob)

    with tf.variable_scope("decoding", reuse=True) as decoding_scope:
        infer_logits = decoding_layer_infer(
            encoder_state, dec_cell, dec_embeddings,
            target_vocab_to_int['<GO>'], target_vocab_to_int['<EOS>'],
          sequence_length, vocab_size, decoding_scope, output_fn, keep_prob)

    return train_logits, infer_logits

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_decoding_layer(decoding_layer)
def seq2seq_model(input_data, target_data, keep_prob, batch_size, sequence_
length, source_vocab_size,
                  target_vocab_size, enc_embedding_size, dec_embedding_size,
rnn_size, num_layers,
                  target_vocab_to_int):
    """
    Build the Sequence-to-Sequence part of the neural network
    :param input_data: Input placeholder
    :param target_data: Target placeholder
    :param keep_prob: Dropout keep probability placeholder
    :param batch_size: Batch Size
    :param sequence_length: Sequence Length
    :param source_vocab_size: Source vocabulary size
    :param target_vocab_size: Target vocabulary size
    :param enc_embedding_size: Decoder embedding size
```

```
    :param dec_embedding_size: Encoder embedding size
    :param rnn_size: RNN Size
    :param num_layers: Number of layers
    :param target_vocab_to_int: Dictionary to go from the target words to an
id
    :return: Tuple of (Training Logits, Inference Logits)
    """
    # TODO: Implement Function
    enc_embed_input = tf.contrib.layers.embed_sequence(
        input_data, source_vocab_size, enc_embedding_size)
    enc_state = encoding_layer(
        enc_embed_input, rnn_size, num_layers, keep_prob)

    proc_target_data = process_decoding_input(target_data, target_vocab_to_
int, batch_size)
    dec_embeddings = tf.Variable(tf.random_uniform([target_vocab_size, dec_
embedding_size]))
     dec_embed_input = tf.nn.embedding_lookup(dec_embeddings, proc_target_
data)

    train_logits, refer_logits = decoding_layer(
        dec_embed_input, dec_embeddings, enc_state, target_vocab_size,
      sequence_length, rnn_size, num_layers, target_vocab_to_int, keep_prob)

    return train_logits, refer_logits

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_seq2seq_model(seq2seq_model)
# Number of Epochs
epochs = 10
# Batch Size
batch_size = 512
# RNN Size
rnn_size = 512
# Number of Layers
num_layers = 4
# Embedding Size
encoding_embedding_size = 128
decoding_embedding_size = 128
# Learning Rate
learning_rate = 0.001
# Dropout Keep Probability
keep_probability = 0.75
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
save_path = 'checkpoints/dev'
(source_int_text, target_int_text), (source_vocab_to_int, target_vocab_to_
int), _ = helper.load_preprocess()
max_source_sentence_length = max([len(sentence) for sentence in source_int_
text])
```

```
train_graph = tf.Graph()
with train_graph.as_default():
    input_data, targets, lr, keep_prob = model_inputs()
   sequence_length = tf.placeholder_with_default(max_source_sentence_length,
None, name='sequence_length')
    input_shape = tf.shape(input_data)

    train_logits, inference_logits = seq2seq_model(
            tf.reverse(input_data, [-1]), targets, keep_prob, batch_size,
sequence_length, len(source_vocab_to_int), len(target_vocab_to_int),
          encoding_embedding_size, decoding_embedding_size, rnn_size, num_
layers, target_vocab_to_int)
    tf.identity(inference_logits, 'logits')
    with tf.name_scope("optimization"):
        # Loss function
        cost = tf.contrib.seq2seq.sequence_loss(
            train_logits,
            targets,
            tf.ones([input_shape[0], sequence_length]))
        # Optimizer
        optimizer = tf.train.AdamOptimizer(lr)
        # Gradient Clipping
        gradients = optimizer.compute_gradients(cost)
        capped_gradients = [(tf.clip_by_value(grad, -1., 1.), var) for grad,
var in gradients if grad is not None]
        train_op = optimizer.apply_gradients(capped_gradients)
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import time
def get_accuracy(target, logits):
    """
    Calculate accuracy
    """
    max_seq = max(target.shape[1], logits.shape[1])
    if max_seq - target.shape[1]:
        target = np.pad(
            target,
            [(0,0),(0,max_seq - target.shape[1])],
            'constant')
    if max_seq - logits.shape[1]:
        logits = np.pad(
            logits,
            [(0,0),(0,max_seq - logits.shape[1]), (0,0)],
            'constant')
    return np.mean(np.equal(target, np.argmax(logits, 2)))
train_source = source_int_text[batch_size:]
train_target = target_int_text[batch_size:]
valid_source = helper.pad_sentence_batch(source_int_text[:batch_size])
valid_target = helper.pad_sentence_batch(target_int_text[:batch_size])
with tf.Session(graph=train_graph) as sess:
    sess.run(tf.global_variables_initializer())
    for epoch_i in range(epochs):
```

```
        for batch_i, (source_batch, target_batch) in enumerate(
                helper.batch_data(train_source, train_target, batch_size)):
            start_time = time.time()

            _, loss = sess.run(
                [train_op, cost],
                {input_data: source_batch,
                 targets: target_batch,
                 lr: learning_rate,
                 sequence_length: target_batch.shape[1],
                 keep_prob: keep_probability})

            batch_train_logits = sess.run(
                inference_logits,
                {input_data: source_batch, keep_prob: 1.0})
            batch_valid_logits = sess.run(
                inference_logits,
                {input_data: valid_source, keep_prob: 1.0})

            train_acc = get_accuracy(target_batch, batch_train_logits)
              valid_acc = get_accuracy(np.array(valid_target), batch_valid_
logits)
            end_time = time.time()
              print('Epoch {:>3} Batch {:>4}/{} - Train Accuracy: {:>6.3f},
Validation Accuracy: {:>6.3f}, Loss: {:>6.3f}'
                .format(epoch_i, batch_i, len(source_int_text) // batch_size,
train_acc, valid_acc, loss))
    # Save Model
    saver = tf.train.Saver()
    saver.save(sess, save_path)
    print('Model Trained and Saved')
helper.save_params(save_path)
import tensorflow as tf
import numpy as np
import helper
import problem_unittests as tests
_, (source_vocab_to_int, target_vocab_to_int), (source_int_to_vocab, target_
int_to_vocab) = helper.load_preprocess()
load_path = helper.load_params()
def sentence_to_seq(sentence, vocab_to_int):
    """
    Convert a sentence to a sequence of ids
    :param sentence: String
    :param vocab_to_int: Dictionary to go from the words to an id
    :return: List of word ids
    """
    # TODO: Implement Function
    sentence_int = list()

    for word in sentence.lower().split(' '):
        try:
            word_int = vocab_to_int[word]
        except:
```

```
            word_int = vocab_to_int['<UNK>']
        sentence_int.append(word_int)
    return sentence_int

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_sentence_to_seq(sentence_to_seq)
translate_sentence = 'he saw a old yellow truck .'

"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
translate_sentence = sentence_to_seq(translate_sentence, source_vocab_to_
int)
loaded_graph = tf.Graph()
with tf.Session(graph=loaded_graph) as sess:
    # Load saved model
    loader = tf.train.import_meta_graph(load_path + '.meta')
    loader.restore(sess, load_path)
    input_data = loaded_graph.get_tensor_by_name('input:0')
    logits = loaded_graph.get_tensor_by_name('logits:0')
    keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')
     translate_logits = sess.run(logits, {input_data: [translate_sentence],
keep_prob: 1.0})[0]
print('Input')
print('  Word Ids:      {}'.format([i for i in translate_sentence]))
print(' English Words: {}'.format([source_int_to_vocab[i] for i in translate_
sentence]))
print('\nPrediction')
print('  Word Ids:       {}'.format([i for i in np.argmax(translate_logits,
1)]))
print('    French  Words:  {}'.format([target_int_to_vocab[i]  for  i  in
np.argmax(translate_logits, 1)]))
english = ''
french = ''
for i in translate_sentence:
    english += source_int_to_vocab[i] + ' '

for i in np.argmax(translate_logits, 1):
    if i != 1:
        french += target_int_to_vocab[i] + ' '

print('English sentence   : ' + english)
print('French translation : ' + french)
```

RNN exemplo básico de geração de TV

A utilização de uma *Deep Learning* baseada em regressão voltada para um conjunto de dados da TV é bem legal. O link do código fonte é https://github.com/paulovpcotta/tv-script-generation, e o conjunto de dados é da Udacity.com.

```
import helper
data_dir = './data/simpsons/moes_tavern_lines.txt'
text = helper.load_data(data_dir)
# Ignore notice, since we don't use it for analysing the data
text = text[81:]
view_sentence_range = (0, 10)
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
import numpy as np
print('Dataset Stats')
print('Roughly the number of unique words: {}'.format(len({word: None for
word in text.split()})))
scenes = text.split('\n\n')
print('Number of scenes: {}'.format(len(scenes)))
sentence_count_scene = [scene.count('\n') for scene in scenes]
print('Average number of sentences in each scene: {}'.format(np.
average(sentence_count_scene)))
sentences = [sentence for scene in scenes for sentence in scene.split('\n')]
print('Number of lines: {}'.format(len(sentences)))
word_count_sentence = [len(sentence.split()) for sentence in sentences]
print('Average number of words in each line: {}'.format(np.average(word_
count_sentence)))
print()
print('The sentences {} to {}:'.format(*view_sentence_range))
print('\n'.join(text.split('\n')[view_sentence_range[0]:view_sentence_
range[1]]))
import numpy as np
import problem_unittests as tests
from collections import Counter
def create_lookup_tables(text):
    """
    Create lookup tables for vocabulary
    :param text: The text of tv scripts split into words
    :return: A tuple of dicts (vocab_to_int, int_to_vocab)
    """
    counts = Counter(text)
    vocab = sorted(counts, key = counts.get, reverse = True)
    vocab_to_int = {word: i for i, word in enumerate(vocab, 0)}
    int_to_vocab = dict(enumerate(vocab, 0))
    return vocab_to_int, int_to_vocab

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_create_lookup_tables(create_lookup_tables)
```

```
def token_lookup():
    """
    Generate a dict to turn punctuation into a token.
     :return: Tokenize dictionary where the key is the punctuation and the
value is the token
    """
    dict = {
        '.' : '||period||',
        ',' : '||comma||',
        '"' : '||quotation_mark||',
        ';' : '||semicolon||',
        '!' : '||esclamation_mark||',
        '?' : '||question_mark||',
        '(' : '||left_parentheses||',
        ')' : '||right_parentheses||',
        '--': '||dash||',
        '\n': '||return||'
    }
        return dict
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_tokenize(token_lookup)
helper.preprocess_and_save_data(data_dir,    token_lookup,    create_lookup_
tables)
import helper
import numpy as np
import problem_unittests as tests
int_text, vocab_to_int, int_to_vocab, token_dict = helper.load_preprocess()
from distutils.version import LooseVersion
import warnings
import tensorflow as tf
# Check TensorFlow Version
assert  LooseVersion(tf.__version__)  >=  LooseVersion('1.0'),  'Please  use
TensorFlow version 1.0 or newer'
print('TensorFlow Version: {}'.format(tf.__version__))
# Check for a GPU
if not tf.test.gpu_device_name():
     warnings.warn('No  GPU  found.  Please  use  a  GPU  to  train  your  neural
network.')
else:
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
def get_inputs():
    """
    Create TF Placeholders for input, targets, and learning rate.
    :return: Tuple (input, targets, learning rate)
    """
    input = tf.placeholder(tf.int32, [None, None], name='input')
    targets = tf.placeholder(tf.int32, [None, None], name='targets')
    learning_rate = tf.placeholder(tf.float32, None, name='learning_rate')
    return input, targets, learning_rate

"""
```

```
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_get_inputs(get_inputs)
def get_init_cell(batch_size, rnn_size):
    """
    Create an RNN Cell and initialize it.
    :param batch_size: Size of batches
    :param rnn_size: Size of RNNs
    :return: Tuple (cell, initialize state)
    """

    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
    cell = tf.contrib.rnn.MultiRNNCell([lstm])
      initial_state = tf.identity(cell.zero_state(batch_size, tf.float32),
name='initial_state')
    return cell, initial_state

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_get_init_cell(get_init_cell)
def get_embed(input_data, vocab_size, embed_dim):
    """
    Create embedding for <input_data>.
    :param input_data: TF placeholder for text input.
    :param vocab_size: Number of words in vocabulary.
    :param embed_dim: Number of embedding dimensions
    :return: Embedded input.
    """

    # From the skip-gram exercise:
    #    For the embedding matrix, I suggest you initialize it with a uniform
    #    random numbers between -1 and 1 using tf.random_uniform.

     embedding = tf.Variable(tf.random_uniform((vocab_size, embed_dim), -1,
1))
    embed = tf.nn.embedding_lookup(embedding, input_data)

    return embed

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_get_embed(get_embed)
def build_rnn(cell, inputs):
    """
    Create a RNN using a RNN Cell
    :param cell: RNN Cell
    :param inputs: Input text data
    :return: Tuple (Outputs, Final State)
    """
    outputs, state = tf.nn.dynamic_rnn(cell, inputs, dtype=tf.float32)
    final_state = tf.identity(state, name='final_state')
```

```
    return outputs, final_state

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_build_rnn(build_rnn)
def build_nn(cell, rnn_size, input_data, vocab_size):
    """
    Build part of the neural network
    :param cell: RNN cell
    :param rnn_size: Size of rnns
    :param input_data: Input data
    :param vocab_size: Vocabulary size
    :return: Tuple (Logits, FinalState)
    """

    embedded = get_embed(input_data, vocab_size, rnn_size)
    outputs, final_state = build_rnn(cell, embedded)
    logits = tf.contrib.layers.fully_connected(
        inputs = outputs,\
        num_outputs = vocab_size,\
        activation_fn = None,\
        weights_initializer=tf.truncated_normal_initializer(stddev= 0.1),\
        biases_initializer=tf.zeros_initializer())
    return logits, final_state

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_build_nn(build_nn)
def get_batches(int_text, batch_size, seq_length):
    """
    Return batches of input and target
    :param int_text: Text with the words replaced by their ids
    :param batch_size: The size of batch
    :param seq_length: The length of sequence
    :return: Batches as a Numpy array
    """
    n_batches = (len(int_text) - 1) // (batch_size * seq_length)
    batches = np.zeros((n_batches, 2, batch_size, seq_length))
    for b in range(n_batches):
        for j in range(batch_size):
            batches[b][0][j] = int_text[seq_length*(n_batches*j+b)    : seq_
length*(n_batches*j+b+1)]
            batches[b][1][j] = int_text[seq_length*(n_batches*j+b)+1 : seq_
length*(n_batches*j+b+1)+1]

    return batches

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_get_batches(get_batches)
```

```
Number of Epochs
num_epochs = 200
# Batch Size
batch_size = 256
# RNN Size
rnn_size = 256
# Sequence Length
seq_length = 16
# Learning Rate
learning_rate = 0.01
# Show stats for every n number of batches
show_every_n_batches = 16
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
save_dir = './save'
from tensorflow.contrib import seq2seq
train_graph = tf.Graph()
with train_graph.as_default():
    vocab_size = len(int_to_vocab)
    input_text, targets, lr = get_inputs()
    input_data_shape = tf.shape(input_text)
    cell, initial_state = get_init_cell(input_data_shape[0], rnn_size)
    logits, final_state = build_nn(cell, rnn_size, input_text, vocab_size)
    # Probabilities for generating words
    probs = tf.nn.softmax(logits, name='probs')
    # Loss function
    cost = seq2seq.sequence_loss(
        logits,
        targets,
        tf.ones([input_data_shape[0], input_data_shape[1]]))
    # Optimizer
    optimizer = tf.train.AdamOptimizer(lr)
    # Gradient Clipping
    gradients = optimizer.compute_gradients(cost)
    capped_gradients = [(tf.clip_by_value(grad, -1., 1.), var) for grad, var
in gradients]
    train_op = optimizer.apply_gradients(capped_gradients)
batches = get_batches(int_text, batch_size, seq_length)
with tf.Session(graph=train_graph) as sess:
    sess.run(tf.global_variables_initializer())
    for epoch_i in range(num_epochs):
        state = sess.run(initial_state, {input_text: batches[0][0]})
        for batch_i, (x, y) in enumerate(batches):
            feed = {
                input_text: x,
                targets: y,
                initial_state: state,
                lr: learning_rate}
          train_loss, state, _ = sess.run([cost, final_state, train_op], feed)
            # Show every <show_every_n_batches> batches
           if (epoch_i * len(batches) + batch_i) % show_every_n_batches == 0:
              print('Epoch {:>3} Batch {:>4}/{}   train_loss = {:.3f}'.format(
```

```
                    epoch_i,
                    batch_i,
                    len(batches),
                    train_loss))
    # Save Model
    saver = tf.train.Saver()
    saver.save(sess, save_dir)
    print('Model Trained and Saved')
helper.save_params((seq_length, save_dir))
import tensorflow as tf
import numpy as np
import helper
import problem_unittests as tests
_, vocab_to_int, int_to_vocab, token_dict = helper.load_preprocess()
seq_length, load_dir = helper.load_params()
def get_tensors(loaded_graph):
    """
     Get input, initial state, final state, and probabilities tensor from
<loaded_graph>
    :param loaded_graph: TensorFlow graph loaded from file
      :return: Tuple (InputTensor, InitialStateTensor, FinalStateTensor,
ProbsTensor)
    """
    input = loaded_graph.get_tensor_by_name("input:0")
    initial_state = loaded_graph.get_tensor_by_name("initial_state:0")
    final_state = loaded_graph.get_tensor_by_name("final_state:0")
    probs = loaded_graph.get_tensor_by_name("probs:0")

    return input, initial_state, final_state, probs

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_get_tensors(get_tensors)
def pick_word(probabilities, int_to_vocab):
    """
    Pick the next word in the generated text
    :param probabilities: Probabilites of the next word
    :param int_to_vocab: Dictionary of word ids as the keys and words as the
values
    :return: String of the predicted word
    """

    #Makes a cumulative sum of the probability list
    t = np.cumsum(probabilities)
    # Select random float from 0 to 1.
    rand_s = np.sum(probabilities) * np.random.rand(1)
   # Select a random word from the list using the probability   distribution
    pred_word = int_to_vocab[int(np.searchsorted(t, rand_s))]
    return pred_word
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
```

```
tests.test_pick_word(pick_word)
gen_length = 200
# homer_simpson, moe_szyslak, or Barney_Gumble
prime_word = 'moe_szyslak'
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
loaded_graph = tf.Graph()
with tf.Session(graph=loaded_graph) as sess:
    # Load saved model
    loader = tf.train.import_meta_graph(load_dir + '.meta')
    loader.restore(sess, load_dir)
    # Get Tensors from loaded model
    input_text, initial_state, final_state, probs = get_tensors(loaded_graph)
    # Sentences generation setup
    gen_sentences = [prime_word + ':']
    prev_state = sess.run(initial_state, {input_text: np.array([[1]])})
    # Generate sentences
    for n in range(gen_length):
        # Dynamic Input
          dyn_input = [[vocab_to_int[word] for word in gen_sentences[-seq_
length:]]]
        dyn_seq_length = len(dyn_input[0])
        # Get Prediction
        probabilities, prev_state = sess.run(
            [probs, final_state],
            {input_text: dyn_input, initial_state: prev_state})

        pred_word = pick_word(probabilities[dyn_seq_length-1], int_to_vocab)
        gen_sentences.append(pred_word)

    # Remove tokens
    tv_script = ' '.join(gen_sentences)
    for key, token in token_dict.items():
        ending = ' ' if key in ['\n', '(', '"'] else ''
        tv_script = tv_script.replace(' ' + token.lower(), key)
    tv_script = tv_script.replace('\n ', '\n')
    tv_script = tv_script.replace('( ', '(')

    print(tv_script)
```

Processo de classificação e predição com Pokémon

É um trabalho realizado no GitHub https://github.com/paulovpcotta/pokemon-udacity-final, que se baseia em classificar e predizer qual pokémon ganhará a batalha. Nesse caso, o conjunto de dados utilizado foi o do Kaggle https://www.kaggle.com/terminus7/pokemon-challenge/kernels.

```
import numpy as np
import pandas as pd
import sklearn
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
pokemon = pd.read_csv('pokemon.csv')
# Impressão de alguns dados do DataSet pokemon.csv
pokemon.head()
# Verificar typer e NaNs
pokemon.info()
pd.value_counts(pokemon['Legendary'])
# Combate dos pokémons
combat = pd.read_csv('combats.csv')
combat.head()
# Combate
cols = ["First_pokemon","Second_pokemon","Winner"]
new_combat_data=combat[cols].replace(pokemon.Name)
new_combat_data.head()
# Teste Winner
# Troca o true e false por números booleans 0 = false e 1 = true
combat.Winner[combat.Winner == combat.First_pokemon] = 0
combat.Winner[combat.Winner == combat.Second_pokemon] = 1
# Normalização dos dados
########################################
# Processo de normalização dos dados a partir das colunas definidas
########################################
def normalization(data_df):
    stats=["HP","Attack","Defense","Sp. Atk","Sp. Def","Speed","Legendary"]
    stats_df=pokemon[stats].T.to_dict("list")
    one=data_df.First_pokemon.map(stats_df)
    two=data_df.Second_pokemon.map(stats_df)
    temp_list=[]
    for i in range(len(one)):
        temp_list.append(np.array(one[i])-np.array(two[i]))
    new_test = pd.DataFrame(temp_list, columns=stats)
    for c in stats:
        description=new_test[c].describe()
           new_test[c]=(new_test[c]-description['min'])/(description['max']-
description['min'])
    return new_test
# Normaliza
data = normalization(combat)
data = pd.concat([data,combat.Winner], axis=1)
# Evitar as colunas NaNs
data.dropna().head()
data.dropna(axis=1).head()
# Evitar as colunas NaNs
# Trata as mesmas com zero
data['HP']=data['HP'].fillna(0)
data['Attack']=data['Attack'].fillna(0)
data['Defense']=data['Defense'].fillna(0)
data['Sp. Atk']=data['Sp. Atk'].fillna(0)
data['Sp. Def']=data['Sp. Def'].fillna(0)
data['Speed']=data['Speed'].fillna(0)
data['Legendary']=data['Legendary'].fillna(0)
data['Winner']=data['Winner'].fillna(0)
print(data)
# Labels
x_label = data.drop("Winner",axis=1)
```

```
y_label = data[“Winner”]
# Treino
from sklearn.model_selection import train_test_split
# Tratamento de NaN
np.isnan(x_label.any())
np.isnan(x_label.any())
# Tratamento de Finite
np.isfinite(y_label.all())
np.isfinite(y_label.all())
x_train, x_test, y_train, y_test = train_test_split(x_label, y_label, test_
size=0.25, random_state=42)
# Saída da Acurácia do treino
print('Versão do SKLearn {}.'.format(sklearn.__version__))
clf = RandomForestClassifier(n_estimators=100)
model = clf.fit(x_train, y_train)
pred = model.predict(x_test)
print('Acurácia de = ', accuracy_score(pred, y_test)*100) # Teste inicial 65%
de acurácia
# Testes
test_data=pd.read_csv('tests.csv')
test_data.head()
# Verificando os NaNs
test_data.info()
test_data['First_pokemon'] = test_data['First_pokemon'].fillna(0)
test_data['Second_pokemon'] = test_data['Second_pokemon'].fillna(0)
new_test_data=test_data[[“First_pokemon”,”Second_pokemon”]].
replace(pokemon.Name)
new_test_data['First_pokemon'] = new_test_data['First_pokemon'].fillna(0)
new_test_data['Second_pokemon'] = new_test_data['Second_pokemon'].fillna(0)
new_test_data.head()
# Processo preditivo normalização
final_data = normalization(test_data)
final_data.head()
final_data.info()
final_data['HP'] = final_data['HP'].fillna(0)
final_data['Attack'] = final_data['Attack'].fillna(0)
final_data['Defense'] = final_data['Defense'].fillna(0)
final_data['Sp. Atk'] = final_data['Sp. Atk'].fillna(0)
final_data['Sp. Def'] = final_data['Sp. Def'].fillna(0)
final_data['Speed'] = final_data['Speed'].fillna(0)
final_data['Legendary'] = final_data['Legendary'].fillna(0)
# Preditivo
pred = model.predict(final_data)
# Processo de avaliação dos dados e direcionar qual pokémon é o vencedor
test_data[“Winner”] = [test_data[“First_pokemon”][i] if pred[i]==0 else test_
data[“Second_pokemon”][i] for i in range(len(pred))]
# Avaliando vencedores
import random
combats_name = test_data[cols].replace(pokemon.Name)
# Recupero o e faço um random para impressão dos dados dos pokémons
print(random.randint(0, len(combats_name)))
combats_name[0:random.randint(0,total)]
```

REFERÊNCIAS

"Google's AlphaGo AI wins three-match series against the world's best Go player". TechCrunch. , 25 Maio maio 2017.

ACKLEY, D. H.; HINTON, G. E.; SEJNOWSKI, T. J. ***A learning algorithm for boltzmann machines.*** Cognitive science, Elsevier, 1985.

AIZENBERG, Igor Aizenberg,; AIZENBERG, Naum N. Aizenberg,; JOOS, P.L. ***Vandewalle. Multi-Valued and Universal Binary Neurons:*** *Theory, Learning and Applications*. Springer Science & Business Media, 2000.

BAKER, J.; DENG, Li; GLASS, Jim; KHUDANPUR, S.; LEE, C.-H.; MORGAN, N.; O'SHAUGHNESSY, D. **"*Research Developments and Directions in Speech Recognition and Understanding*",** 2009.

Balázs Csanád Csáji. ***Approximation with Artificial Neural Networks.;*** Faculty of Sciences; Eötvös Loránd University, Hungary, 2011.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. ***Representation Learning:*** *A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013.*

BENGIO, Yoshua; LECUN, Yann; HINTON, Geoffrey. ***"Deep Learning"***. Nature, 2015.

BOREN, W. L.;, HUNTER, T. B.;, BJELLAND, J. C.;, HUNT, K. R. ***Comparison of breast consistency at palpation with breast density at mammography.*** Invest Radiol, 1990.

DENG, L.; YU, D. *"**Deep Learning: Methods and Applications**"* **(PDF).** *Foundations and Trends in Signal Processing*, 2014.

HINTON, G. E.; SRIVASTAVA, N.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. R. ***"Improving neural networks by preventing co-adaptation of feature detectors"***, 2012.

HINTON, Geoffrey E.; DAYAN, Peter; FREY, Brendan J.; NEAL, Radford. ***"The wake-sleep algorithm for unsupervised neural networks"***, 1995.

HORNIK, Kurt. *"**Approximation Capabilities of Multilayer Feedforward Networks**"*. Neural Networks, 1991.

KAPUR, Lenny. ***Neural Networks & The Backpropagation Algorithm, Explained.*** 1th ed. Spring, 2010.

KRIZHEVSKY; SUTSKEVER; HINTON. ***Methods research image.*** 2012.

LECUN *et al.*, *"**Backpropagation Applied to Handwritten Zip Code Recognition**,"*, Neural Computation, 1989.

LU, Z., .; PU, H., .; WANG, F., .; HU, Z., .; & WANG, L. ***The Expressive Power of Neural Networks:*** *A View from the Width*, 2017.

LUI, Bing *et al*. ***Lifelong Machine Learning:*** *Second Edition*. 2018.

MORGAN, Nelson; BOURLARD, Hervé; RENALS, Steve; COHEN, Michael; FRANCO, Horacio. ***"Hybrid neural network/hidden markov model systems for continuous speech recognition"***. *International Journal of Pattern Recognition and Artificial Intelligence*, 1998.

NASCIMENTO, Rodrigo. **Afinal, o que é Big Data?**. Disponível em: <http://marketingpordados.com/analise-de-dados/o-que-e-big-data-%F0%9F%A4%96/>. Publicado em: 2017. Acesso em: 17 de jun. de 2019.

OLSHAUSEN, B. A. ***"Emergence of simple-cell receptive field properties by learning a sparse code for natural images"***. Nature, 1996.

SCHMIDHUBER, J. ***"Deep Learning in Neural Networks: An Overview".*** Neural Networks, 2015.

SHAVLIK, J. W., and ***Dietterich, T. G. Readings in Machine Learning.*** San Mateo, CA: Morgan Kaufmann Publishers, 1990.

WAIBEL, A.; HANAZAWA, T.; HINTON, G.; SHIKANO, K.; LANG, K. J. ***"Phoneme recognition using time-delay neural networks"***, 1989.

WENG, J. WENG, ; AHUJA, N. AHUJA AND; HUANG, T. S. *HUANG,* ***"Learning recognition and segmentation using the Cresceptron",*** 1997.