No Jupyter Notebook vc executa uma célula com shift-enter

Vamos usar uma rede neural muito simples via a framework keras para fazer uma tarefa de classificação. O problema que vamos tratar é bastante conhecido da comunidade de machine learning, e vai servir principalmente para nós concretizarmos o que foi visto em sala. É importante que tenham o ambiente de execução configurado, como foi sugerido no começo do curso. Nossa tarefa é projetar uma rede neural mlp de forma que ela consiga reconhecer dígitos escritos. Keras já vem com uma base de datos destes dígitos para treinamento, ou seja, um tensor com (imagens de dígitos, identificaçao do dígito) chamada de mnist. Portanto a primeira coisa é importar no nosso código o keras (façam isso onde a conexão de internet for boa)

In [105…
```
from keras.datasets import mnist
import keras
import numpy as np
import os

os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

A primeira etapa é obtermos os dados, como já foi dito o próprio keras tem estes dados que são muito usados por quem pratica machine learning. Os dados são de treinamento da rede e de teste da rede

In [106…
```
(train_images,train_labels),(test_images,test_labels) = mnist.load_data()
```

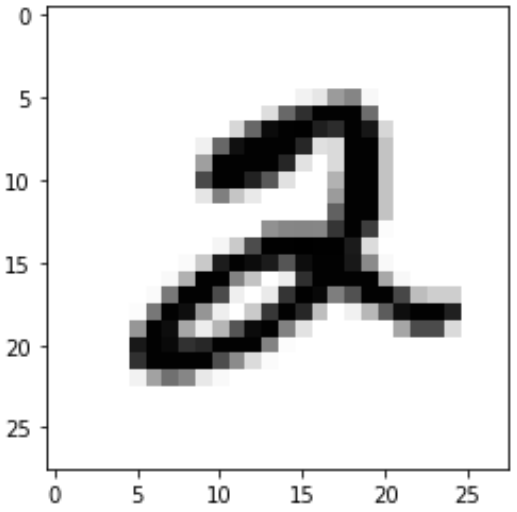
Estes dados já estão bem arrumadinhos. Evidentemente quando estivermos lidando com dados reais há toda uma etapa de pre-processamento onde vamos verificar dados faltosos, dados fora de escala, dados errados etc.. Vamos ver o jeitão destes dados.

In [107…
```
print(train_images.shape)
```

```
(60000, 28, 28)
```

O primeiro componente mostra que de treino temos 60000 exemplos, cada um deles numa matriz de 28 x 28 (que é a imagem). Vamos olhar um, plotando a imagem , o sexto exemplo:

In [108…
```
digito = train_images[5]
%matplotlib inline
import matplotlib.pyplot as plt
plt.imshow(digito, cmap = plt.cm.binary)
plt.show()
```



Redes neurais lidam melhor com dados reais, e não inteiros como é o caso aqui. Vamos converter num formato mais adequado ao processamento de redes

In [109…
```
train_images = train_images.reshape((60000,28*28))
train_images = train_images.astype('float32')/255
test_image_copy = test_images
test_images = test_images.reshape((10000,28*28))
test_images = test_images.astype('float32')/255
```

Agora vem a questão do projeto da rede neural. Decidimos que nosso modelo de machine learning será uma rede neural padrão (mlp). O que precisamos ainda decidir é o número de camadas que esta rede terá, e como serão os neurônios desta rede. Como isso teremos fechada a questão da arquitetura da rede. Nossa rede terá todos os neurônios conectados aos da próxima camada, e todas as sinapses serão ajustadas numa camada, ou seja, uma típica MLP. Quem faz isso no keras é o modelo chamado sequencial.

In [110…
```
from keras import models
from keras import layers

rede = models.Sequential()
```

Agora é a questão da arquitetura propriamente dita : quantas camadas (layers) quantos neurônios por camada, que função de ativação estes neurônios vão usar..

In [111…
```
rede.add(layers.Dense(512,activation = 'sigmoid', input_shape = (28*28,) ))
rede.add(layers.Dense(10,activation='softmax'))
```

Então decidi por 1 camadas. A primeira (hidden) terá 512 neurônios com relu como função de ativação. E como é a primeira, portanto conectada ao input, informo como é um dado de entrada padrão : é 28x28 A segunda camada é a de saída, ela tem 10 neurônios pois a tarefa é discriminar, identificar uma imagem 28x28 como sendo um entre os dígitos de 0 a 9 . A última camada usa uma função de ativação do tipo softmax, esta função de ativação funciona como uma probabilidade, ou seja, a rede vai estar respondendo qual é a probabilidade de que a imagem vista é o número n. Como são 10 números , haverá 10 probabilidades e a maior é a resposta da rede.

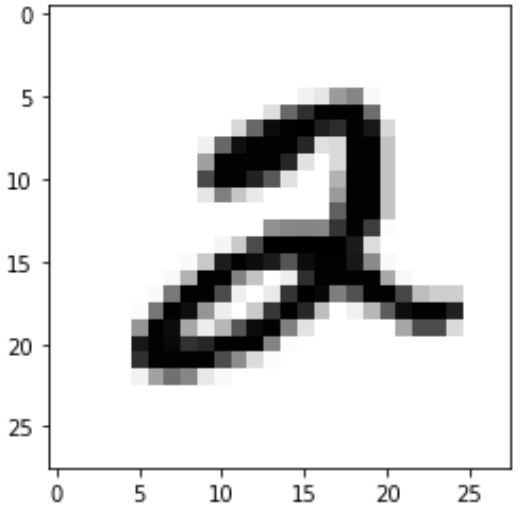
Arquitetura preparada, agora temos que cuidar do esquema de aprendizado, ou treinamento. Vimos que se usa uma função custo. Uma função que vai indicar se a rede está se saindo bem num dado, ou mal. Derivamos o algoritmo backpropagation usando uma função custo que era o erro médio quadrático da rede. Para usar o keras com rede nossa função custo será a função loss, a mais adequada para tarefas de classificação como a nossa é a de cross-entropia categorica. loss = 'categorical_crossentropy' E evidentemente precisamos de um "processo" através do qual usar esta informação da função custo para ajustar os pesos. Derivamos o famoso backpropagation em sala, que nada mais é que um cálculo de gradiente. Estamos então diante de um processo de otimização: achar as melhores sinapses que minimizam uma função custo. Em keras configuramos que otimizador usar. E temos que indicar também como otimizar, qual medida é importante. A métrica

In [112…
```
rede.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Falta um detalhe, precisamos tratar também as respostas. Decidimos que a rede vai respoder probabilidades, mas os dados de treino tem outra forma : (28x28, digito) Digito, nos dados , é um valor único. Mas a rede espera que sejam 10! A saída dela tem 10 neurônios, não 1

In [113…
```
plt.imshow(digito, cmap = plt.cm.binary)
print('nos dados  , para esta imagem, a resposta é ',train_labels[5])
```

```
nos dados  , para esta imagem, a resposta é  2
```



Isso é fácil de resolver no nosso caso, keras ajuda também. Vamos transformar os labels de dígitos para categorias

In [114…
```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

In [115…
```
print('Agora nosso 2, virou ', train_labels[5])
```

```
Agora nosso 2, virou  [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

Que é o que a rede espera, 10 respostas para seus 10 neurônios na última camada.

Resta-nos preparar o treinamento da rede. Vamos empregar um esquema de treinamento conhecido com batch learning. No batch learning a rede não processa todo o conjunto de treinamento de uma vez, ela quebra em subconjuntos chamados batchs. Nossa rede vai usar batchs de tamanho 128. Vamos treinar só por 5 épocas, onde uma época é o correspondente a apresentação de todos os dados de treino

In [116…
```
treino = rede.fit(train_images, train_labels, epochs=20, batch_size=128)
```

```
Epoch 1/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.4211 - accur
acy: 0.8851
Epoch 2/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.2346 - accur
acy: 0.9319
Epoch 3/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.1776 - accur
acy: 0.9483
Epoch 4/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.1395 - accur
acy: 0.9593
Epoch 5/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1125 - accur
acy: 0.9675
Epoch 6/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0937 - accur
acy: 0.9725
Epoch 7/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.0787 - accur
acy: 0.9764
Epoch 8/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0673 - accur
acy: 0.9806
Epoch 9/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0582 - accur
acy: 0.9828
Epoch 10/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0503 - accur
acy: 0.9852
Epoch 11/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0440 - accur
acy: 0.9875
Epoch 12/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0382 - accur
acy: 0.9888
Epoch 13/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0334 - accur
acy: 0.9908
Epoch 14/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0290 - accur
acy: 0.9924
Epoch 15/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0252 - accur
acy: 0.9933
Epoch 16/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0221 - accur
acy: 0.9944
Epoch 17/20
```

```
60000/60000 [==============================] - 5s 82us/step - loss: 0.0189 - accur
acy: 0.9955
Epoch 18/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0165 - accur
acy: 0.9961
Epoch 19/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0143 - accur
acy: 0.9965
Epoch 20/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0124 - accur
acy: 0.9973
```

A rede foi treinada! Esperemos... O resultado do treino, no keras, fica num objeto chamado history. É um dicionário { chave, valor }

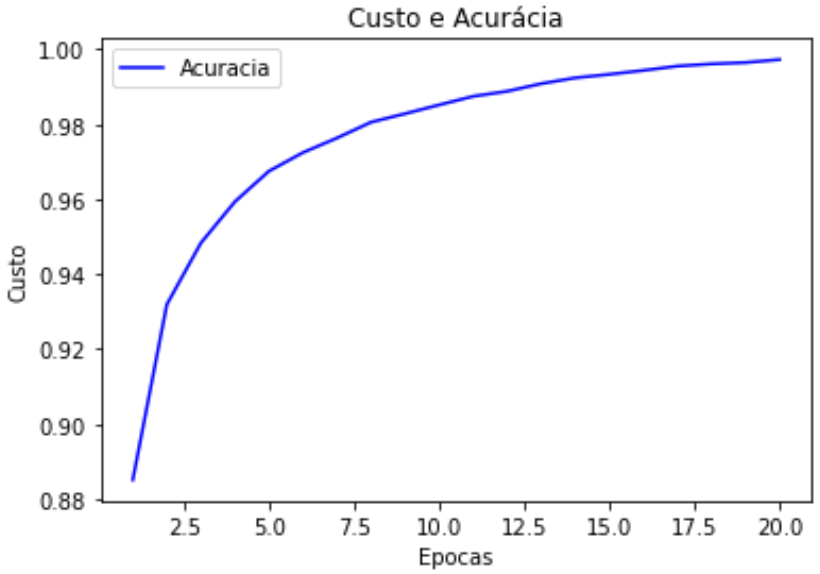
In [117…
```
historia = treino.history
```

In [118…
```
print(historia.keys())
```

```
dict_keys(['loss', 'accuracy'])
```

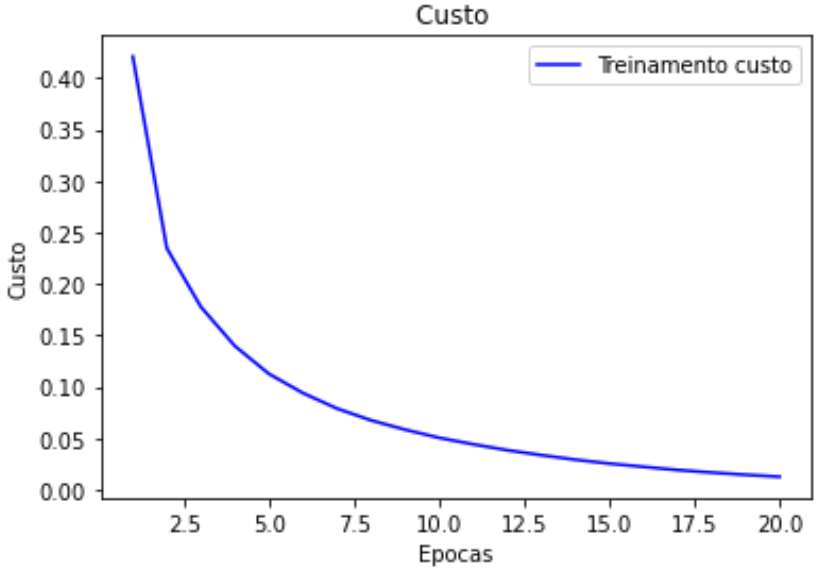
O objeto historia do treino tem 2 elementos : acuracia e custo. Vamos plotar :

In [119…
```
import matplotlib.pyplot as plt

acc = treino.history['accuracy']

loss = treino.history['loss']

epochs = range(1, len(acc) + 1)

# "bo" ponto azul
#plt.plot(epochs, loss, 'bo', label='Treinamento custo')
# b linha azul"
plt.plot(epochs, acc, 'b', label='Acuracia')
plt.title('Custo e Acurácia')
plt.xlabel('Epocas')
plt.ylabel('Custo')
plt.legend()

plt.show()
```



In [120…

```
loss = treino.history['loss']

epochs = range(1, len(acc) + 1)

# "bo" ponto azul
plt.plot(epochs, loss, 'b', label='Treinamento custo')
# b linha azul"
#plt.plot(epochs, acc, 'b', label='Acuracia')
plt.title('Custo ')
plt.xlabel('Epocas')
plt.ylabel('Custo')
plt.legend()

plt.show()
```



Temos uma rede que aprendeu a reconhecer dígitos numa imagem. Agora ela pode ser usada em dados novos, nunca vistos por ela antes. Os dados de teste.

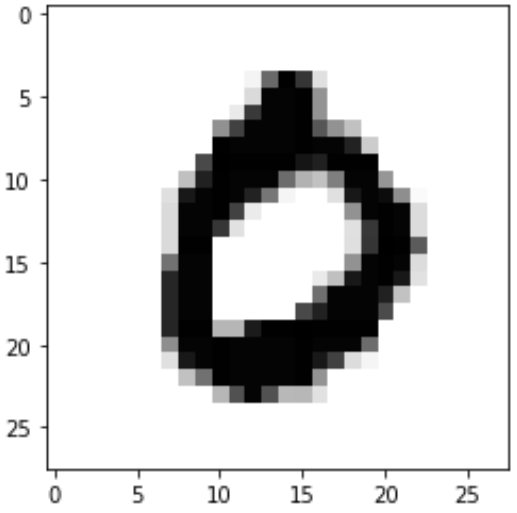
In [121…

```
respostas = rede.predict(test_images )
```

Para checar um dígito qualquer, vamos por a imagem de teste na forma que tínhamos no começo.

In [126…
```
imagem_escolhida = 3
```

In [127…
```
digito = test_image_copy[imagem_escolhida]
%matplotlib inline
import matplotlib.pyplot as plt
plt.imshow(digito, cmap = plt.cm.binary)
plt.show()
```



In [128…
```
print(respostas[imagem_escolhida])
```

```
[9.9996746e-01 3.8903036e-09 1.4916031e-06 4.6784862e-07 8.0728512e-07
 5.7675913e-07 7.2864605e-06 3.9566753e-06 3.0544194e-09 1.7926612e-05]
```

In [125…
```
print('A maior probabilidade é: ',np.amax(respostas[imagem_escolhida]))
print('Imagem corresponde ao número: ', np.argmax(respostas[imagem_escolhida]))
```

```
A maior probabilidade é:  0.999841
Imagem corresponde ao número:  9
```

TAREFAS : reduza o número de épocas, retreine a rede e discuta a acurácia nos dados de treino e de teste aumente muito o número de epocas (500), retreine, e discuta a acurácia nos dados de treino e teste. Modifique a arquitetura (mais camadas, mais neuronios por camada) retreine e discuta a acurácia nos dados de treino e teste. Entregar até 27/05 como relatório Vale um ponto na média

In [ ]:
```
```