MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Módulo III - Arquiteturas de CNNS e treinamento de redes profundas

Exercícios com soluções

Moacir Antonelli Ponti

CeMEAI - ICMC/USP São Carlos

Recomenda-se fortemente que os exercícios sejam feitos sem consultar as respostas antecipadamente.

Exercício 1)

Considere 4 funções de custo distintas: 1. entropia cruzada binária, 2. perda quadrática, vistas em aula, e mais duas adicionais:

1. Perda 0-1

$$\frac{1}{N}\sum_{i=1}^{N}\begin{cases}0 & \text{if } y_i = \hat{y}_i \\ 1 & \text{if } y_i \neq \hat{y}_i\end{cases}$$

1. Perda SVM/Hinge

$$\frac{1}{N}\sum_{i=1}^{N}\max(0, 1 - y_i^h \cdot f(x_i)),$$

essa função considera que as classes são -1 e 1, sendo $f(x_i) = \hat{y}_i^h$ um valor de saída considerando valores negativos (os quais gerarão classificação para a classe -1) e positivos (classificação para a classe 1). Portanto será preciso adaptar as classes do problema e a saída $\hat{y}^h$ para esse cenário da seguinte forma:

- $y^h \in \{-1, 1\}$, e
- $\hat{y}^h = 2 \cdot (\hat{y} - 0.5)$, sendo $\hat{y}$ a probabilidade de uma instância pertencer à classe positiva (1).

Considere o exemplo dado em aula, com os pontos unidimensionais conforme o

código abaixo.

A seguir, treine um classificador de Regressão Logística com solver `lbfgs` e compute as quatro perdas nesse conjunto de dados após o treinamento. Note que as perdas 1,2 e 4 são calculadas com base nas probabilidades, enquanto que 3 é calculada com base na classificação.

Imprima as perdas por instância para inspeção e logo após a perda média no conjunto de treinamento. Qual a ordem de magnitude das perdas, da menor para a maior?

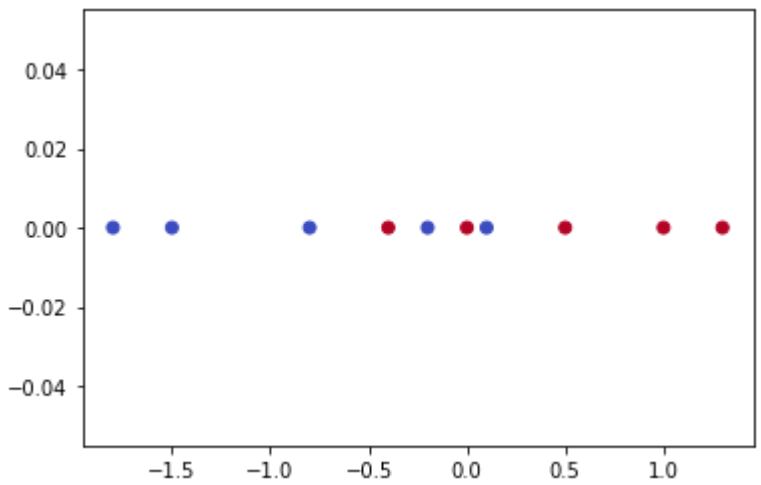
(a) Hinge, Quadrática, Entropia Cruzada, 0-1
(b) Quadrática, Entropia Cruzada, Hinge e 0-1
(c) 0-1, Quadrática, Entropia Cruzada, Hinge
(d) Quadrática, 0-1, Entropia Cruzada, Hinge

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([-1.8,-1.5,-0.8,-0.4,-0.2, 0.0, 0.1, 0.5, 1.0, 1.3])
y = np.array([ 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0,  1.0, 1.0, 1.0])
yh = np.array([ -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.

plt.scatter(x,np.zeros(10), c=y,cmap=plt.cm.coolwarm)
```

Out[1]:
```
<matplotlib.collections.PathCollection at 0x7f4c366cb1f0>
```



In [2]:
```
from sklearn.linear_model import LogisticRegression

# treinando o modelo
logr1 = LogisticRegression(solver='lbfgs')
logr1.fit(x.reshape(-1, 1), y)

# pegando as probabilidades de saída
y_hat = logr1.predict_proba(x.reshape(-1, 1))[:, 1].ravel()
print('y1    = {}'.format(np.round(y , 3)))
print('y_hat = {}'.format(np.round(y_hat, 3)))

# classificando para calcular a perda 0-1
y_clas = y_hat.copy()
y_clas[y_clas>=0.5] = 1
y_clas[y_clas<0.5] = 0

# calculando a saída hinge, entre -1 e 1
y_hat_hi = (y_hat-0.5)*2
print('y_hat_hi= {}'.format(np.round(y_hat_hi, 3)))

# perda quadrática
loss_qu = np.power(y-y_hat,2)
# perda de entropia cruzada
loss_ec = -(y*np.log(y_hat+.00001) + (1-y)*np.log(1-y_hat +.00001))
# perda zero-um
loss_01 = (y!=y_clas)*1

# perda hinge
hi_mult = 1-(yh*y_hat_hi)
loss_hi = [max(0,mi) for mi in hi_mult]
print('1-y*y_hat= {}'.format(np.round(hi_mult, 3)))
```

```
y1    = [0. 0. 0. 1. 0. 1. 0. 1. 1. 1.]
y_hat = [0.133 0.178 0.325 0.433 0.49  0.547 0.576 0.682 0.792 0.84
3]
y_hat_hi= [-0.735 -0.645 -0.349 -0.134 -0.02   0.095  0.151  0.365
0.585  0.687]
1-y*y_hat= [0.265 0.355 0.651 1.134 0.98  0.905 1.151 0.635 0.415 0.
313]
```

In [3]:
```
print("Perdas calculadas por instância:")
print(np.round(loss_qu,3))
print(np.round(loss_ec,3))
print(np.round(loss_01,3))
print(np.round(loss_hi,3))
print()

print("Perda quadrática = %.4f" % (np.mean(loss_qu)))
print("Entropia cruzada = %.4f" % (np.mean(loss_ec)))
print("Perda 0-1        = %.4f" % (np.mean(loss_01)))
print("Perda hinge/svm  = %.4f" % (np.mean(loss_hi)))
```

```
Perdas calculadas por instância:
[0.018 0.032 0.106 0.321 0.24  0.205 0.331 0.101 0.043 0.025]
[0.142 0.195 0.394 0.837 0.674 0.603 0.857 0.382 0.233 0.17 ]
[0 0 0 1 0 0 1 0 0 0]
[0.265 0.355 0.651 1.134 0.98  0.905 1.151 0.635 0.415 0.313]
```

```
Perda quadrática = 0.1421
Entropia cruzada = 0.4487
Perda 0-1        = 0.2000
Perda hinge/svm  = 0.6805
```

Exercício 2)

Considere as funções de custo vistas em aula e estudadas no exercício anterior: Perda Quadrática (MSE), Erro Absoluto (MAE), Perda 0-1, Perda Hinge/SVM, Entropia Cruzada. Como escolher uma função para realizar o treinamento de uma rede neural?

(a) Na dúvida escolher sempre a entropia cruzada, pois é a mais popular e considerada um padrão na literatura da área de redes neurais
(b) Desde que a função permita medir o erro do modelo atual, permite por consequência também medir o custo de escolher os parâmetros atuais, então qualquer função pode ser utilizada sem restrições
(c) Considerar o problema em questão: classificação binária, multiclasse, regressão, etc e entender a magnitude dos valores das funções com base no problema e sua capacidade de guiar o modelo no processo de convergência
(d) Em geral, a entropia cruzada deve ser utilizada para problemas de classificação, e a perda quadrática para problemas de regressão, não sendo necessário investigar outras funções de custo pois são mais relevantes outros parâmetros como a taxa de aprendizado e o tamanho do batch

Justificativa: Apesar de mais popular, a entropia cruzada pode não funcionar bem em todos os cenários. Por outro lado, fazer uma busca exaustiva é impraticável. Assim, é preciso tomar uma decisão "educada" com base no problema em questão, os valores de saída, e selecionar um subconjunto de funções candidatas a serem investigadas para resolver o problema.

Exercício 3)

Considerando as funções de perda: entropia cruzada categórica e perda quadrática, qual é o valor das perdas para um exemplo arbitrário no momento da inicialização aleatória de um modelo numa tarefa de classificação de 5 classes?

(a) Entropia Cruzada = 1.6; Quadrática = 0.8
(b) Entropia Cruzada = 2.3; Quadrática = 0.8
(c) Entropia Cruzada = 1.6; Quadrática = 0.16
(d) Entropia Cruzada = 0.32; Quadrática = 0.8

Justificativa: veja código abaixo. Na inicialização aleatória, teríamos um classificador gerando um vetor de probabilidade com a distribuição aproximadamente uniforme, ou seja, todos os valores 0.2=1/5. Computando a entropia cruzada categórica, temos apenas o -log do valor predito para a classe verdadeira, enquanto que na quadrática, a soma dos erros cometidos ao longo do vetor

In [4]:
```
y = np.array([.0, .0, .0, .0, 1.0])
yh = np.array([0.2, 0.2, 0.2, 0.2, 0.2])

loss_ec = -np.sum((y*np.log(yh+.00001)))

loss_qu = np.sum(np.power(y-yh,2))
print(loss_ec)
print(loss_qu)
```

```
1.6093879136840585
0.8000000000000002
```

Exercício 4)

Sobre os métodos de otimização, o que podemos dizer quando comparamos SGD e Adam?

(a) Ambos realizam atualização iterativa dos parâmetros usando o gradiente, mas o Adam incorpora mecanismos baseados em gradientes anteriores, e o segundo momento do gradiente como ponderação da taxa de aprendizado
(a) O SGD é equivalente ao Adam quando aplicado Momentum no algoritmo SGD base
(c) Ambos realizam atualização iterativa dos parâmetros usando o gradiente, mas apenas SGD permite decaimento da taxa de aprendizado
(d) O Adam é um algoritmo de otimização que obtém sempre melhores resultados do que o SGD e suas variações

Justificativa: o SGD utiliza apenas o gradiente, enquanto o Adam computa primeiro momento do gradiente corrigida, e o segundo momento como forma de ponderar a magnitude do passo. As outras alternativas são inválidas porque: Adam utiliza uma estratégia similar, mas não igual ao momentum, e também possui um tipo de taxa de aprendizado adaptativa; Adam também permite decaimento da taxa de aprendizado; finalmente, não é possível dizer que um algoritmo de otimização é melhor sempre. Ainda que tenha mecanismos mais sofisticados, notar por exemplo que muitos modelos do estado da arte são treinados com SGD + Momentum.

Exercício 5)

Dentre as alternativas, escolha a prática válida mais relevante ao projetar o treinamento de redes profundas

(a) Inicializar todos os pesos com valores aleatórios e utilizar o maior número de instâncias possíveis no treinamento, garantindo que os hiperparâmetros com valor padrão obterão bons resultados
(b) Utilizar sempre a função de custo entropia cruzada, para a qual é recomendado o uso do otimizador Adam e taxa de aprendizado com decaimento. Definir a melhor taxa de decaimento de forma a minimizar a diferença entre o custo de treinamento e validação
(c) Utilizar conjunto pequeno de instâncias para busca grosseira de hiperparâmetros como: otimizador, taxa de aprendizado, momentum e tamanho de batch, e depois refinar a busca num conjunto maior com base em métricas obtidas nos conjuntos de validação e treinamento
(d) Rezar para Yan LeCun, Yoshua Bengio, Geoffrey Hinton e Kunihiko Fukushima.

Justificativa: nem sempre os valores padrão serão bons hiperparâmetros. Ainda que algumas escolhas sejam populares (como uso de Adam e Entropia Cruzada), o melhor é sempre realizar uma busca, ainda que grosseira com poucos dados, por parâmetros que se ajustem à arquitetura projetada. Se você acredita, rezar pode até te acalmar, mas não vai ajudar no treinamento da rede. Felizmente os 4 estão vivos, então tentar contatá-los no Twitter pode ser uma opção :)

Exercício 6)

Qual a principal diferença das arquiteturas VGGNet, Inception e Residual Network com relação à suas camadas convolucionais?

(a) A VGGNet possui camadas convolucionais com filtros de mesmo tamanho 3×3, enquanto as outras arquiteturas, Inception e ResNet aplicam filtros 5×5 ou com concatenação de mapas de ativação ao longo da rede
(b) A rede Inception permite treinamento com maior número de camadas quanto comparada à VGGNet, que por sua vez permite treinamento com maior número de camadas quanto comparada à ResNet
(c) A VGGNet possui camadas convolucionais sequenciais, eventualmente seguidas de MaxPooling, enquanto a ResNet computa mapas de ativação de com diferentes filtros, concatenando-os, e a Inception possui um módulo do tipo banco de filtros, que permite saltar para camadas futuras, facilitando o treinamento com mais camadas
(d) A VGGNet possui camadas convolucionais sequenciais, enquanto Inception possui camadas convolucionais paralelas, e ResNet tem mapas de ativação que desviam da lógica sequencial e pulam camadas

Justificativa: Sua principal diferença é o fluxo durante a rede, sendo a VGG sequencial e as outras duas cujas ativações dão saltos (ResNet) ou possuem paralelismo (Inception). Alternativa (a) está errada pois ResNet não aplica filtros de tamanho maior do que 5x5, nem realiza concatenação de mapas de ativação (mas sim a soma); (b) é inválida pois a ResNet permite treinar com mais camadas do que a VGG; (c) está errada pois Inception não possui saltos nas camadas, nem ResNet possui concatenação de mapas. 1

Exercício 7)

Utilizando a biblioteca Keras, investige o impacto do uso de parâmetros padrão de learning rate na base de dados Boston Housing, com relação ao uso de decaimento de learning rate, a partir de um valor estabelecido.

Carregue a base de dados e normalize os atributos com z-score. Crie uma rede com camadas densas: 16, 8 e 1 (de saída), todas com ativação `relu` .

Treine por 50 épocas 2 redes neurais com a função de custo `mse` , medindo também a `mae`

A. Com otimizador Adam e todos os outros parâmetros no valor padrão B. Com otimizador Adam, iniciando com learning rate 0.02 e decaimento exponencial de 0.05 a partir da época 10

Antes de projetar, compilar e treinar cada rede, defina as sementes do numpy para 1 e do tensorflow para 2.

Use os dados de teste como "validação" durante o treinamento.

Considerando os valores de erro (MSE e MAE) na última época de ambos A e B, e considerando ainda generalização como a divergência entre os erros de treinamento e validação, podemos dizer que:

(a) B obteve menores valores de erro (MSE e MAE) do que A, mas em termos de generalização (entre treinamento e validação) ambos tiveram comportamento similar, com pequena vantagem para B
(b) B obteve menores valores de erro (MSE e MAE) do que A, porém A obteve melhor generalização dos erros (entre treinamento e validação).
(c) B obteve valores de MSE significativamente menores do que A, mas A generalizou melhor com relação ao MAE, indicando que A pdoeria ser treinado por mais épocas.
(d) B obteve MSE de validação próximo a 7, A obteve 21 na mesma métrica, indicando que A não convergiu e apenas B generalizou

Justificativa: Notar no código abaixo que A obteve no treinamento mse: 21 e mae:

3, e na validacao mse: 28 e mae: 4; enquanto isso B obteve no treinamento mse: 7, mae: 2, e na validacao mse: 23, mae:3, assim B foi melhor em geral, mas em termos de generalização A apresentou valores mais próximos quando comparamos treinamento e validação.

In [32]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import boston_housing
from numpy.random import seed
from tensorflow.random import set_seed

(x_train, y_train), (x_test, y_test) = boston_housing.load_data()

mean = x_train.mean(axis=0)
x_train -= mean
std = x_train.std(axis=0)
x_train /= std

x_test -= mean
x_test /= std
```

In [33]:
```
seed(1)
set_seed(2)

model1 = keras.Sequential()
model1.add(keras.layers.Dense(16, activation="relu", input_shape=(x_
model1.add(keras.layers.Dense(8, activation="relu"))
model1.add(keras.layers.Dense(1, activation="relu"))
model1.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 16)                224
_________________________________________________________________
dense_42 (Dense)             (None, 8)                 136
_________________________________________________________________
dense_43 (Dense)             (None, 1)                 9
=================================================================
Total params: 369
Trainable params: 369
Non-trainable params: 0
_________________________________________________________________
```

In [34]:
```
epochs=50
```

In [35]:
```
model1.compile(optimizer="adam",
               loss='mse', metrics=['mae'])

history = model1.fit(x_train, y_train, epochs=epochs,
                     validation_data=(x_test,y_test), verbose=1)
```

```
Epoch 1/50
13/13 [==============================] - 0s 5ms/step - loss: 585.175
4 - mae: 22.3727 - val_loss: 613.9129 - val_mae: 23.0327
Epoch 2/50
13/13 [==============================] - 0s 1ms/step - loss: 583.280
6 - mae: 22.3219 - val_loss: 611.5378 - val_mae: 22.9651
Epoch 3/50
13/13 [==============================] - 0s 1ms/step - loss: 580.066
0 - mae: 22.2220 - val_loss: 607.6867 - val_mae: 22.8512
Epoch 4/50
13/13 [==============================] - 0s 2ms/step - loss: 575.492
1 - mae: 22.0783 - val_loss: 602.3406 - val_mae: 22.6769
Epoch 5/50
13/13 [==============================] - 0s 2ms/step - loss: 569.194
5 - mae: 21.8791 - val_loss: 595.4497 - val_mae: 22.4521
Epoch 6/50
13/13 [==============================] - 0s 2ms/step - loss: 560.904
5 - mae: 21.6187 - val_loss: 586.7144 - val_mae: 22.2004
Epoch 7/50
13/13 [==============================] - 0s 2ms/step - loss: 550.866
7 - mae: 21.3266 - val_loss: 576.0062 - val_mae: 21.8998
Epoch 8/50
13/13 [==============================] - 0s 2ms/step - loss: 539.381
6 - mae: 20.9878 - val_loss: 564.3648 - val_mae: 21.5709
Epoch 9/50
13/13 [==============================] - 0s 2ms/step - loss: 527.277
7 - mae: 20.6267 - val_loss: 552.4490 - val_mae: 21.2194
Epoch 10/50
13/13 [==============================] - 0s 1ms/step - loss: 514.643
1 - mae: 20.2411 - val_loss: 539.1353 - val_mae: 20.8213
Epoch 11/50
13/13 [==============================] - 0s 1ms/step - loss: 500.547
3 - mae: 19.8228 - val_loss: 524.1532 - val_mae: 20.4014
Epoch 12/50
13/13 [==============================] - 0s 1ms/step - loss: 484.504
3 - mae: 19.3594 - val_loss: 506.8472 - val_mae: 19.9354
Epoch 13/50
13/13 [==============================] - 0s 2ms/step - loss: 466.391
1 - mae: 18.8561 - val_loss: 486.8180 - val_mae: 19.4548
Epoch 14/50
13/13 [==============================] - 0s 2ms/step - loss: 445.168
4 - mae: 18.2758 - val_loss: 464.1213 - val_mae: 18.9416
Epoch 15/50
13/13 [==============================] - 0s 1ms/step - loss: 421.757
7 - mae: 17.6592 - val_loss: 438.3264 - val_mae: 18.3936
Epoch 16/50
13/13 [==============================] - 0s 1ms/step - loss: 396.141
6 - mae: 16.9767 - val_loss: 409.8697 - val_mae: 17.7608
Epoch 17/50
13/13 [==============================] - 0s 1ms/step - loss: 368.308
0 - mae: 16.2548 - val_loss: 379.9591 - val_mae: 17.0779
Epoch 18/50
13/13 [==============================] - 0s 1ms/step - loss: 338.560
1 - mae: 15.4908 - val_loss: 349.1888 - val_mae: 16.3332
```

```
Epoch 19/50
13/13 [==============================] - 0s 1ms/step - loss: 308.652
2 - mae: 14.7313 - val_loss: 316.1881 - val_mae: 15.5137
Epoch 20/50
13/13 [==============================] - 0s 1ms/step - loss: 276.849
5 - mae: 13.9183 - val_loss: 283.0538 - val_mae: 14.6546
Epoch 21/50
13/13 [==============================] - 0s 2ms/step - loss: 245.320
7 - mae: 13.0655 - val_loss: 249.0259 - val_mae: 13.6994
Epoch 22/50
13/13 [==============================] - 0s 2ms/step - loss: 213.280
3 - mae: 12.1205 - val_loss: 215.9491 - val_mae: 12.6860
Epoch 23/50
13/13 [==============================] - 0s 2ms/step - loss: 182.300
2 - mae: 11.1449 - val_loss: 185.0045 - val_mae: 11.6367
Epoch 24/50
13/13 [==============================] - 0s 2ms/step - loss: 154.247
6 - mae: 10.1345 - val_loss: 155.4173 - val_mae: 10.6212
Epoch 25/50
13/13 [==============================] - 0s 2ms/step - loss: 127.469
6 - mae: 9.1332 - val_loss: 129.5807 - val_mae: 9.6613
Epoch 26/50
13/13 [==============================] - 0s 2ms/step - loss: 105.905
6 - mae: 8.1996 - val_loss: 107.1700 - val_mae: 8.7351
Epoch 27/50
13/13 [==============================] - 0s 2ms/step - loss: 86.8243
- mae: 7.3650 - val_loss: 90.3774 - val_mae: 7.9372
Epoch 28/50
13/13 [==============================] - 0s 2ms/step - loss: 72.9218
- mae: 6.6659 - val_loss: 76.8644 - val_mae: 7.2161
Epoch 29/50
13/13 [==============================] - 0s 2ms/step - loss: 61.6419
- mae: 6.0603 - val_loss: 67.2146 - val_mae: 6.6441
Epoch 30/50
13/13 [==============================] - 0s 2ms/step - loss: 53.8190
- mae: 5.5992 - val_loss: 60.0445 - val_mae: 6.1753
Epoch 31/50
13/13 [==============================] - 0s 2ms/step - loss: 48.0302
- mae: 5.2448 - val_loss: 54.5258 - val_mae: 5.7935
Epoch 32/50
13/13 [==============================] - 0s 2ms/step - loss: 43.3715
- mae: 4.9288 - val_loss: 50.5551 - val_mae: 5.5295
Epoch 33/50
13/13 [==============================] - 0s 2ms/step - loss: 39.9749
- mae: 4.6981 - val_loss: 47.2896 - val_mae: 5.3326
Epoch 34/50
13/13 [==============================] - 0s 2ms/step - loss: 37.2680
- mae: 4.5086 - val_loss: 44.5110 - val_mae: 5.1706
Epoch 35/50
13/13 [==============================] - 0s 2ms/step - loss: 34.9643
- mae: 4.3443 - val_loss: 42.2012 - val_mae: 5.0323
Epoch 36/50
13/13 [==============================] - 0s 2ms/step - loss: 32.9622
- mae: 4.2131 - val_loss: 40.3673 - val_mae: 4.9243
```

```
Epoch 37/50
13/13 [==============================] - 0s 2ms/step - loss: 31.3310
- mae: 4.1031 - val_loss: 38.7247 - val_mae: 4.8193
Epoch 38/50
13/13 [==============================] - 0s 2ms/step - loss: 29.9206
- mae: 4.0018 - val_loss: 37.2963 - val_mae: 4.7320
Epoch 39/50
13/13 [==============================] - 0s 2ms/step - loss: 28.7498
- mae: 3.9154 - val_loss: 36.0101 - val_mae: 4.6548
Epoch 40/50
13/13 [==============================] - 0s 2ms/step - loss: 27.6514
- mae: 3.8355 - val_loss: 34.8849 - val_mae: 4.5832
Epoch 41/50
13/13 [==============================] - 0s 2ms/step - loss: 26.7511
- mae: 3.7695 - val_loss: 33.8301 - val_mae: 4.5160
Epoch 42/50
13/13 [==============================] - 0s 1ms/step - loss: 25.8317
- mae: 3.7048 - val_loss: 32.9782 - val_mae: 4.4604
Epoch 43/50
13/13 [==============================] - 0s 1ms/step - loss: 25.1418
- mae: 3.6562 - val_loss: 32.1475 - val_mae: 4.4058
Epoch 44/50
13/13 [==============================] - 0s 1ms/step - loss: 24.4319
- mae: 3.6026 - val_loss: 31.2991 - val_mae: 4.3470
Epoch 45/50
13/13 [==============================] - 0s 2ms/step - loss: 23.7944
- mae: 3.5532 - val_loss: 30.6563 - val_mae: 4.3044
Epoch 46/50
13/13 [==============================] - 0s 1ms/step - loss: 23.2665
- mae: 3.5159 - val_loss: 30.0919 - val_mae: 4.2672
Epoch 47/50
13/13 [==============================] - 0s 1ms/step - loss: 22.7724
- mae: 3.4818 - val_loss: 29.4772 - val_mae: 4.2283
Epoch 48/50
13/13 [==============================] - 0s 1ms/step - loss: 22.3118
- mae: 3.4576 - val_loss: 29.0922 - val_mae: 4.2028
Epoch 49/50
13/13 [==============================] - 0s 1ms/step - loss: 21.8989
- mae: 3.4324 - val_loss: 28.6235 - val_mae: 4.1684
Epoch 50/50
13/13 [==============================] - 0s 1ms/step - loss: 21.5179
```

In [36]:

```
seed(1)
set_seed(2)

model2 = keras.Sequential()
model2.add(keras.layers.Dense(16, activation="relu", input_shape=(x_
model2.add(keras.layers.Dense(8, activation="relu"))
model2.add(keras.layers.Dense(1, activation="relu"))
model2.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
dense_44 (Dense)             (None, 16)                224
_________________________________________________________________
dense_45 (Dense)             (None, 8)                 136
_________________________________________________________________
dense_46 (Dense)             (None, 1)                 9
=================================================================
Total params: 369
Trainable params: 369
Non-trainable params: 0
```

In [37]:
```
def scheduler(epoch, lr):
    print("Taxa atual = %.5f" % (lr))
    if epoch < 10:
        return lr
    else:
        return np.round(lr * tf.math.exp(-0.05),4)

callbacklr = keras.callbacks.LearningRateScheduler(scheduler)

model2.compile(optimizer=keras.optimizers.Adam(0.02),
              loss='mse', metrics=['mae'])

history = model2.fit(x_train, y_train, epochs=epochs,
                     callbacks=[callbacklr],  validation_data=(x_tes
                    verbose=1)
```

```
Taxa atual = 0.02000
Epoch 1/50
13/13 [==============================] - 0s 5ms/step - loss: 546.303
3 - mae: 21.0637 - val_loss: 502.7121 - val_mae: 19.6967
Taxa atual = 0.02000
Epoch 2/50
13/13 [==============================] - 0s 1ms/step - loss: 364.738
1 - mae: 16.3605 - val_loss: 209.1097 - val_mae: 12.1302
Taxa atual = 0.02000
Epoch 3/50
13/13 [==============================] - 0s 1ms/step - loss: 88.3731
- mae: 7.2638 - val_loss: 78.3675 - val_mae: 6.6225
Taxa atual = 0.02000
Epoch 4/50
13/13 [==============================] - 0s 1ms/step - loss: 41.4213
- mae: 4.5843 - val_loss: 40.2851 - val_mae: 5.1418
Taxa atual = 0.02000
Epoch 5/50
13/13 [==============================] - 0s 1ms/step - loss: 28.1764
- mae: 4.0251 - val_loss: 28.4457 - val_mae: 4.1727
Taxa atual = 0.02000
Epoch 6/50
13/13 [==============================] - 0s 2ms/step - loss: 21.8347
- mae: 3.3420 - val_loss: 24.9516 - val_mae: 3.9363
Taxa atual = 0.02000
Epoch 7/50
13/13 [==============================] - 0s 2ms/step - loss: 19.0849
- mae: 3.2354 - val_loss: 26.3385 - val_mae: 3.9648
Taxa atual = 0.02000
```

```
Epoch 8/50
13/13 [==============================] - 0s 2ms/step - loss: 17.1346
- mae: 3.0490 - val_loss: 25.4212 - val_mae: 3.7553
Taxa atual = 0.02000
Epoch 9/50
13/13 [==============================] - 0s 1ms/step - loss: 15.2533
- mae: 2.8776 - val_loss: 24.7372 - val_mae: 3.6326
Taxa atual = 0.02000
Epoch 10/50
13/13 [==============================] - 0s 2ms/step - loss: 13.8437
- mae: 2.7122 - val_loss: 25.8824 - val_mae: 3.6150
Taxa atual = 0.02000
Epoch 11/50
13/13 [==============================] - 0s 2ms/step - loss: 12.6995
- mae: 2.6255 - val_loss: 24.6819 - val_mae: 3.4543
Taxa atual = 0.01900
Epoch 12/50
13/13 [==============================] - 0s 1ms/step - loss: 12.2264
- mae: 2.5415 - val_loss: 23.9559 - val_mae: 3.3889
Taxa atual = 0.01810
Epoch 13/50
13/13 [==============================] - 0s 2ms/step - loss: 11.4202
- mae: 2.4677 - val_loss: 25.0335 - val_mae: 3.3759
Taxa atual = 0.01720
Epoch 14/50
13/13 [==============================] - 0s 2ms/step - loss: 10.9026
- mae: 2.4250 - val_loss: 24.0920 - val_mae: 3.3163
Taxa atual = 0.01640
Epoch 15/50
13/13 [==============================] - 0s 2ms/step - loss: 10.9228
- mae: 2.3915 - val_loss: 25.3525 - val_mae: 3.3452
Taxa atual = 0.01560
Epoch 16/50
13/13 [==============================] - 0s 1ms/step - loss: 10.5868
- mae: 2.3339 - val_loss: 22.8815 - val_mae: 3.2061
Taxa atual = 0.01480
Epoch 17/50
13/13 [==============================] - 0s 1ms/step - loss: 10.1835
- mae: 2.3254 - val_loss: 24.8257 - val_mae: 3.2882
Taxa atual = 0.01410
Epoch 18/50
13/13 [==============================] - 0s 1ms/step - loss: 9.7149
- mae: 2.2557 - val_loss: 24.9670 - val_mae: 3.2495
Taxa atual = 0.01340
Epoch 19/50
13/13 [==============================] - 0s 1ms/step - loss: 9.6145
- mae: 2.2383 - val_loss: 24.2749 - val_mae: 3.2392
Taxa atual = 0.01270
Epoch 20/50
13/13 [==============================] - 0s 2ms/step - loss: 9.6776
- mae: 2.2196 - val_loss: 24.1411 - val_mae: 3.2410
Taxa atual = 0.01210
Epoch 21/50
13/13 [==============================] - 0s 2ms/step - loss: 9.1538
```

```
- mae: 2.1823 - val_loss: 25.4822 - val_mae: 3.2217
Taxa atual = 0.01150
Epoch 22/50
13/13 [==============================] - 0s 1ms/step - loss: 9.1131
- mae: 2.1562 - val_loss: 24.0012 - val_mae: 3.1864
Taxa atual = 0.01090
Epoch 23/50
13/13 [==============================] - 0s 2ms/step - loss: 9.0302
- mae: 2.1679 - val_loss: 24.2062 - val_mae: 3.1824
Taxa atual = 0.01040
Epoch 24/50
13/13 [==============================] - ETA: 0s - loss: 6.0845 - ma
e: 1.841 - 0s 2ms/step - loss: 8.8143 - mae: 2.1191 - val_loss: 24.5
451 - val_mae: 3.1889
Taxa atual = 0.00990
Epoch 25/50
13/13 [==============================] - 0s 1ms/step - loss: 8.7356
- mae: 2.1085 - val_loss: 23.9083 - val_mae: 3.1614
Taxa atual = 0.00940
Epoch 26/50
13/13 [==============================] - 0s 1ms/step - loss: 8.6935
- mae: 2.1340 - val_loss: 25.3673 - val_mae: 3.2476
Taxa atual = 0.00890
Epoch 27/50
13/13 [==============================] - 0s 1ms/step - loss: 8.6450
- mae: 2.1152 - val_loss: 24.8235 - val_mae: 3.1728
Taxa atual = 0.00850
Epoch 28/50
13/13 [==============================] - 0s 1ms/step - loss: 8.2366
- mae: 2.0541 - val_loss: 24.2383 - val_mae: 3.1798
Taxa atual = 0.00810
Epoch 29/50
13/13 [==============================] - 0s 2ms/step - loss: 8.1566
- mae: 2.0214 - val_loss: 23.2697 - val_mae: 3.1009
Taxa atual = 0.00770
Epoch 30/50
13/13 [==============================] - 0s 2ms/step - loss: 8.1561
- mae: 2.0512 - val_loss: 23.9299 - val_mae: 3.1568
Taxa atual = 0.00730
Epoch 31/50
13/13 [==============================] - 0s 2ms/step - loss: 8.0103
- mae: 2.0198 - val_loss: 23.9747 - val_mae: 3.1116
Taxa atual = 0.00690
Epoch 32/50
13/13 [==============================] - 0s 2ms/step - loss: 8.0539
- mae: 2.0009 - val_loss: 24.1620 - val_mae: 3.1396
Taxa atual = 0.00660
Epoch 33/50
13/13 [==============================] - 0s 2ms/step - loss: 7.8383
- mae: 2.0024 - val_loss: 23.7048 - val_mae: 3.1033
Taxa atual = 0.00630
Epoch 34/50
13/13 [==============================] - 0s 2ms/step - loss: 7.9419
- mae: 2.0194 - val_loss: 23.4670 - val_mae: 3.0881
```

```
Taxa atual = 0.00600
Epoch 35/50
13/13 [==============================] - 0s 2ms/step - loss: 7.7736
- mae: 1.9632 - val_loss: 23.0987 - val_mae: 3.0756
Taxa atual = 0.00570
Epoch 36/50
13/13 [==============================] - 0s 2ms/step - loss: 7.6836
- mae: 1.9679 - val_loss: 24.5179 - val_mae: 3.1361
Taxa atual = 0.00540
Epoch 37/50
13/13 [==============================] - 0s 2ms/step - loss: 7.7157
- mae: 1.9988 - val_loss: 23.3377 - val_mae: 3.0777
Taxa atual = 0.00510
Epoch 38/50
13/13 [==============================] - 0s 2ms/step - loss: 7.6302
- mae: 1.9679 - val_loss: 23.2640 - val_mae: 3.0926
Taxa atual = 0.00490
Epoch 39/50
13/13 [==============================] - 0s 2ms/step - loss: 7.6193
- mae: 1.9748 - val_loss: 24.0490 - val_mae: 3.0977
Taxa atual = 0.00470
Epoch 40/50
13/13 [==============================] - 0s 2ms/step - loss: 7.5694
- mae: 1.9438 - val_loss: 23.3482 - val_mae: 3.0718
Taxa atual = 0.00450
Epoch 41/50
13/13 [==============================] - 0s 2ms/step - loss: 7.4949
- mae: 1.9470 - val_loss: 23.3571 - val_mae: 3.0678
Taxa atual = 0.00430
Epoch 42/50
13/13 [==============================] - 0s 2ms/step - loss: 7.5751
- mae: 1.9698 - val_loss: 24.0104 - val_mae: 3.1203
Taxa atual = 0.00410
Epoch 43/50
13/13 [==============================] - 0s 2ms/step - loss: 7.5106
- mae: 1.9379 - val_loss: 22.9066 - val_mae: 3.0254
Taxa atual = 0.00390
Epoch 44/50
13/13 [==============================] - 0s 2ms/step - loss: 7.3981
- mae: 1.9424 - val_loss: 23.3354 - val_mae: 3.0664
Taxa atual = 0.00370
Epoch 45/50
13/13 [==============================] - 0s 2ms/step - loss: 7.3400
- mae: 1.9366 - val_loss: 23.5260 - val_mae: 3.0674
Taxa atual = 0.00350
Epoch 46/50
13/13 [==============================] - 0s 2ms/step - loss: 7.3797
- mae: 1.9237 - val_loss: 23.2982 - val_mae: 3.0590
Taxa atual = 0.00330
Epoch 47/50
13/13 [==============================] - 0s 2ms/step - loss: 7.3028
- mae: 1.9392 - val_loss: 23.4771 - val_mae: 3.0693
Taxa atual = 0.00310
Epoch 48/50
```

```
13/13 [==============================] - 0s 2ms/step - loss: 7.3255
- mae: 1.9405 - val_loss: 23.7739 - val_mae: 3.0792
Taxa atual = 0.00290
Epoch 49/50
13/13 [==============================] - 0s 2ms/step - loss: 7.3127
- mae: 1.9238 - val_loss: 23.1459 - val_mae: 3.0436
Taxa atual = 0.00280
```

Exercício 8)

Utilizando ainda a biblioteca Keras, investige o impacto do uso de parâmetros padrão de batchsize na base de dados Boston Housing, agora utilizando a mesma arquitetura da atividade anterior, com otimizador Adam, iniciando com learning rate 0.02 e decaimento exponencial de 0.1 a partir da época 6.

Investige valores de batch = 2, 4, 8, 16, 32, 64, 128 e 256 executando por um número de épocas proporcional ao tamanho do batch atual, calculado por: $epocas = \lfloor \log_2(512 \cdot batchsize) \rfloor$, valor deve ser convertido para inteiro. Esse número de épocas proporcional permite equilibrar um pouco a relação entre velocidade e quantidade de vezes que os parâmetros são adaptados.

Antes de projetar, compilar e treinar cada rede, defina as sementes do numpy para 1 e do tensorflow para 2.

Não passe dados de validação durante o treinamento. Após o treinamento, avalie MSE nos dados de teste e imprima para comparar os valores para diferentes batchsizes.

Quais foram os dois piores e os dois melhores valores de tamanho de batch em termos do MSE de teste?

(a) Piores: 128 e 256; Melhores: 4 e 16
(b) Piores: 16 e 64; Melhores: 32 e 128
(c) Piores: 2 e 4; Melhores: 32 e 64
(d) Piores: 4 e 256; Melhores: 2 e 128

Justificativa: Notar no código abaixo a comparação.

In [41]:
```
def my_dnn():
    model2 = keras.Sequential()
    model2 = keras.Sequential()
    model2.add(keras.layers.Dense(16, activation="relu", input_shape
    model2.add(keras.layers.Dense(8, activation="relu"))
    model2.add(keras.layers.Dense(1, activation="relu"))
    return model2
```

In [81]:
```
def scheduler(epoch, lr):
    if epoch < 6:
        return lr
    else:
        return np.round(lr * tf.math.exp(-0.1),4)

callbacklr = keras.callbacks.LearningRateScheduler(scheduler)
```

In [87]:
```
batches = [2, 4, 8, 16, 32, 64, 128, 256]

for batch_size in batches:
    seed(1)
    set_seed(2)
    model = my_dnn()
    model.compile(optimizer=keras.optimizers.Adam(0.02),
                  loss='mse', metrics=['mae'])

    epochs = int(np.log2((batch_size*512)))
    history = model.fit(x_train, y_train, epochs=epochs, batch_size=
                        callbacks=[callbacklr], verbose=0)

    score = model.evaluate(x_test, y_test, verbose = 0)
    print("Batch size = %d, Erro de Validação MSE = %.4f (Epocas=%d)
```

```
Batch size = 2, Erro de Validação MSE = 22.4658 (Epocas=10)
Batch size = 4, Erro de Validação MSE = 19.1443 (Epocas=11)
Batch size = 8, Erro de Validação MSE = 23.9285 (Epocas=12)
Batch size = 16, Erro de Validação MSE = 21.6256 (Epocas=13)
Batch size = 32, Erro de Validação MSE = 25.3148 (Epocas=14)
Batch size = 64, Erro de Validação MSE = 23.7474 (Epocas=15)
Batch size = 128, Erro de Validação MSE = 30.6913 (Epocas=16)
Batch size = 256, Erro de Validação MSE = 129.7601 (Epocas=17)
```

Exercício 9)

O que podemos concluir dos dois exercícios anteriores (7 e 8)?

(a) Os valores padrão para os hiperparâmetros geram bons resultados. A busca por outros parâmetros pode não valer a pena pois a diferença alcançada observada é pequena.
(b) Devemos sempre utilizar Adam com decaimento de taxa de aprendizado e batch size de tamanho entre 8 e 64, sendo que o uso do padrão (32) é normalmente suficiente.
(c) Batchs de tamanho muito grande são prejudiciais ao treinamento, e o otimizador Adam é sempre melhor com decaimento de taxa de aprendizado.
(d) O uso de hiperparâmetros com valores padrão pode gerar resultados subótimos, sendo importante uma busca de parâmetros para melhor otimizar modelos

---

Exercício 10)

Carregue a base de dados Fashion MNIST

Crie duas redes neurais utilizando os blocos Residuais e módulos Inception conforme visto em aula.

- InceptionNet
  - Módulo Inception V1 com número de filtros: 32, 32, 32, 32, 32, 16
  - Maxpooling com pool=2, stride=2
  - Módulo Inception V1 com número de filtros: 32, 64, 64, 64, 64, 16
  - Maxpooling com pool=2, stride=2
- ResNet
  - 3 blocos residuais com 64 filtros, cada um seguido por camada Maxpooling com pool=2, stride=2

Ambos devem possuir uma camada `GlobalAveragePooling2D` antes da camada de predição.

Treine ambas com SGD, learning rate 0.05 e momentum 0.8, utilizando batchsize 64, e apenas as 800 primeiras imagens do dataset de treinamento (use :800), por 100 épocas.

Ao final compute a perda e a acurácia no treinamento (800 imagens) e teste (todas as imagens do teste), e exiba o gráfico da perda ao longo das épocas para as duas arquiteturas.

Marque a alternativa que melhor se encaixa no resultado observado e sua conclusão.

(a) A ResNet conseguiu ajustar perfeitamente aos dados de treinamento, mas com perda mais alta calculada no teste, indicando overfitting, enquanto a Inception tem espaço para melhorias
(b) As duas arquiteturas obtiveram boa generalização, mas a Inception possui claras vantagens frente à ResNet, com maior acurácia no teste
(c) A Inception teve melhor generalização mas não convergiu para perda próxima a zero com 100 épocas, portanto a taxa de aprendizado escolhida poderia ser reduzida para obter resultados melhores, enquanto a ResNet poderia ser treinada por mais épocas para melhorar a acurácia no teste.
(d) Os resultados de acurácia e perda no teste da ResNet são muito diferentes daqueles obtidos no treinamento, indicando underfitting, ou seja, uma incapacidade do modelo de se ajustar aos dados utilizados no processo de aprendizado

Justificativa: Nenhuma arquitetura apresenta boa generalização, visto que tanto a

perda quanto a acurácia diferem quando computadas no treinamento e no teste. O caso dessa diferença é um sinal de overfitting, e não de underfitting, já que o modelo foi capaz de se ajustar perfeitamente (com perda 0) para o treinamento. Além disso, reduzir a taxa de aprendizado torna a convergência mais lenta, e não mais rápida.

In [14]:
```
# carregando datasets do keras
#from tensorflow.keras.datasets import mnist

from tensorflow.keras.datasets import fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# obtendo informações das imagens (resolucao) e dos rótulos (número
img_lin, img_col = x_train.shape[1], x_train.shape[2]
num_classes = len(np.unique(y_train))

print(x_train.shape)

# dividir por 255 para obter normalizacao
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# transformar categorias em one-hot-encoding
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# verifica imagens da base de dados tem 3 canais (RGB) ou apenas 1 (
if (len(x_train.shape) == 3):
      n_channels = 1
else:
      n_channels = x_train.shape[3]

# re-formata o array de forma a encontrar o formato da entrada (inpu
# se a dimensão dos canais vem primeiro ou após a imagem
if keras.backend.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], n_channels, img_lin,
    x_test = x_test.reshape(x_test.shape[0], n_channels, img_lin, im
    input_shape = (n_channels, img_lin, img_col)
else:
    x_train = x_train.reshape(x_train.shape[0], img_lin, img_col, n_
    x_test = x_test.reshape(x_test.shape[0], img_lin, img_col, n_cha
    input_shape = (img_lin, img_col, n_channels)

print("Shape: ", input_shape)
```

```
(60000, 28, 28)
Shape:  (28, 28, 1)
```

In [15]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import add

def inception_module(layer_in, f1_out, f2_in, f2_out, f3_in, f3_out,
    # 1x1 conv
    conv1 = Conv2D(f1_out, (1,1), padding='same', activation='relu')
    # 3x3 conv
    conv3 = Conv2D(f2_in, (1,1), padding='same', activation='relu')(
    conv3 = Conv2D(f2_out, (3,3), padding='same', activation='relu')
    # 5x5 conv
    conv5 = Conv2D(f3_in, (1,1), padding='same', activation='relu')(
    conv5 = Conv2D(f3_out, (5,5), padding='same', activation='relu')
    # 3x3 max pooling
    pool = MaxPooling2D((3,3), strides=(1,1), padding='same')(layer_
    pool = Conv2D(f4_out, (1,1), padding='same', activation='relu')(
    layer_out = concatenate([conv1, conv3, conv5, pool], axis=-1)
    return layer_out

# define model input
input_layer = Input(shape=input_shape)
# add inception blocks
layer1 = inception_module(input_layer, 32, 32, 32, 32, 32, 16)
pool1 = MaxPooling2D((2,2), strides=(2,2), padding='same')(layer1) #
layer2 = inception_module(pool1, 32, 64, 64, 64, 64, 16) # rem?
pool2 = MaxPooling2D((2,2), strides=(2,2), padding='same')(layer2)
flatt = keras.layers.GlobalAveragePooling2D()(pool2)

softmax = keras.layers.Dense(num_classes, activation='softmax')(flat

# create model
Inception = keras.models.Model(inputs=input_layer, outputs=softmax)
# summarize model
Inception.summary()
```

```
Model: "functional_1"
__________________________________________________________________________
_______________________________
Layer (type)                    Output Shape         Param #     Con
nected to
==========================================================================
==============================
input_1 (InputLayer)            [(None, 28, 28, 1)]  0
__________________________________________________________________________
_______________________________
conv2d_1 (Conv2D)               (None, 28, 28, 32)   64          inp
ut_1[0][0]
__________________________________________________________________________
_______________________________
conv2d_3 (Conv2D)               (None, 28, 28, 32)   64          inp
ut_1[0][0]
__________________________________________________________________________
_______________________________
```

```
max_pooling2d (MaxPooling2D)    (None, 28, 28, 1)    0           inp
ut_1[0][0]
__________________________________________________________________
_____________________________
conv2d (Conv2D)                 (None, 28, 28, 32)   64          inp
ut_1[0][0]
__________________________________________________________________
_____________________________
conv2d_2 (Conv2D)               (None, 28, 28, 32)   9248        con
v2d_1[0][0]
__________________________________________________________________
_____________________________
conv2d_4 (Conv2D)               (None, 28, 28, 32)   25632       con
v2d_3[0][0]
__________________________________________________________________
_____________________________
conv2d_5 (Conv2D)               (None, 28, 28, 16)   32          max
_pooling2d[0][0]
__________________________________________________________________
_____________________________
concatenate (Concatenate)       (None, 28, 28, 112)  0           con
v2d[0][0]
                                                                 con
v2d_2[0][0]
                                                                 con
v2d_4[0][0]
                                                                 con
v2d_5[0][0]
__________________________________________________________________
_____________________________
max_pooling2d_1 (MaxPooling2D)  (None, 14, 14, 112)  0           con
catenate[0][0]
__________________________________________________________________
_____________________________
conv2d_7 (Conv2D)               (None, 14, 14, 64)   7232        max
_pooling2d_1[0][0]
__________________________________________________________________
_____________________________
conv2d_9 (Conv2D)               (None, 14, 14, 64)   7232        max
_pooling2d_1[0][0]
__________________________________________________________________
_____________________________
max_pooling2d_2 (MaxPooling2D)  (None, 14, 14, 112)  0           max
_pooling2d_1[0][0]
__________________________________________________________________
_____________________________
conv2d_6 (Conv2D)               (None, 14, 14, 32)   3616        max
_pooling2d_1[0][0]
__________________________________________________________________
_____________________________
conv2d_8 (Conv2D)               (None, 14, 14, 64)   36928       con
v2d_7[0][0]
__________________________________________________________________
_____________________________
```

```
conv2d_10 (Conv2D)              (None, 14, 14, 64)   102464      con
v2d_9[0][0]
__________________________________________________________________
_____________________________
conv2d_11 (Conv2D)              (None, 14, 14, 16)   1808        max
_pooling2d_2[0][0]
__________________________________________________________________
_____________________________
concatenate_1 (Concatenate)     (None, 14, 14, 176)  0           con
v2d_6[0][0]
                                                                 con
v2d_8[0][0]
                                                                 con
v2d_10[0][0]
                                                                 con
v2d_11[0][0]
__________________________________________________________________
_____________________________
max_pooling2d_3 (MaxPooling2D)  (None, 7, 7, 176)    0           con
catenate_1[0][0]
__________________________________________________________________
_____________________________
global_average_pooling2d (Globa (None, 176)          0           max
_pooling2d_3[0][0]
__________________________________________________________________
_____________________________
dense_30 (Dense)                (None, 10)           1770        glo
bal_average_pooling2d[0][0]
==================================================================
==============================
Total params: 196,154
Trainable params: 196,154
Non-trainable params: 0
```

In [16]:
```
def residual_block(layer_in, n_filters):
    merge_input = layer_in
    #verifica se é necessária uma primeira camada para deixar o núme
    if layer_in.shape[-1] != n_filters:
        merge_input = Conv2D(n_filters, (1,1), padding='same', activ
    # conv1
    conv1 = Conv2D(n_filters, (3,3), padding='same', activation='rel
    # conv2
    conv2 = Conv2D(n_filters, (3,3), padding='same', activation='lin
    # soma entrada com saída (pulou 2 camadas)
    layer_out = add([conv2, merge_input])
    # função de ativação da saída do bloco
    layer_out = keras.layers.Activation('relu')(layer_out)
    return layer_out

# define model input
visible = Input(shape=input_shape)

layer1 = residual_block(visible, 64)
pool1 = MaxPooling2D((2,2), strides=(2,2), padding='same')(layer1)
layer2 = residual_block(pool1, 64)
pool2 = MaxPooling2D((2,2), strides=(2,2), padding='same')(layer2)
layer3 = residual_block(pool2, 64)
pool3 = MaxPooling2D((2,2), strides=(2,2), padding='same')(layer3)
flatt = keras.layers.GlobalAveragePooling2D()(pool3)
softmax = keras.layers.Dense(num_classes, activation='softmax')(flat

# create model
ResNet = keras.models.Model(inputs=visible, outputs=softmax)
# summarize model
ResNet.summary()
```

```
Model: "functional_3"
__________________________________________________________________
_____________________________
Layer (type)                 Output Shape              Param #     Con
nected to
==================================================================
==============================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0
__________________________________________________________________
_____________________________
conv2d_13 (Conv2D)           (None, 28, 28, 64)        640         inp
ut_2[0][0]
__________________________________________________________________
_____________________________
conv2d_14 (Conv2D)           (None, 28, 28, 64)        36928       con
v2d_13[0][0]
__________________________________________________________________
_____________________________
conv2d_12 (Conv2D)           (None, 28, 28, 64)        128         inp
ut_2[0][0]
__________________________________________________________________
_____________________________
add (Add)                    (None, 28, 28, 64)        0           con
v2d_14[0][0]
```

```
                                                                 con
v2d_12[0][0]
__________________________________________________________________________
_____________________________
activation (Activation)         (None, 28, 28, 64)   0           add
[0][0]
__________________________________________________________________________
_____________________________
max_pooling2d_4 (MaxPooling2D)  (None, 14, 14, 64)   0           act
ivation[0][0]
__________________________________________________________________________
_____________________________
conv2d_15 (Conv2D)              (None, 14, 14, 64)   36928       max
_pooling2d_4[0][0]
__________________________________________________________________________
_____________________________
conv2d_16 (Conv2D)              (None, 14, 14, 64)   36928       con
v2d_15[0][0]
__________________________________________________________________________
_____________________________
add_1 (Add)                     (None, 14, 14, 64)   0           con
v2d_16[0][0]
                                                                 max
_pooling2d_4[0][0]
__________________________________________________________________________
_____________________________
activation_1 (Activation)       (None, 14, 14, 64)   0           add
_1[0][0]
__________________________________________________________________________
_____________________________
max_pooling2d_5 (MaxPooling2D)  (None, 7, 7, 64)     0           act
ivation_1[0][0]
__________________________________________________________________________
_____________________________
conv2d_17 (Conv2D)              (None, 7, 7, 64)     36928       max
_pooling2d_5[0][0]
__________________________________________________________________________
_____________________________
conv2d_18 (Conv2D)              (None, 7, 7, 64)     36928       con
v2d_17[0][0]
__________________________________________________________________________
_____________________________
add_2 (Add)                     (None, 7, 7, 64)     0           con
v2d_18[0][0]
                                                                 max
_pooling2d_5[0][0]
__________________________________________________________________________
_____________________________
activation_2 (Activation)       (None, 7, 7, 64)     0           add
_2[0][0]
__________________________________________________________________________
_____________________________
max_pooling2d_6 (MaxPooling2D)  (None, 4, 4, 64)     0           act
ivation_2[0][0]
```

```
_____
global_average_pooling2d_1 (Glo (None, 64)           0           max
_pooling2d_6[0][0]
_____
dense_31 (Dense)                (None, 10)           650         glo
bal_average_pooling2d_1[0][0]
==========
Total params: 186,058
Trainable params: 186,058
Non-trainable params: 0
_____
```

In [17]:
```
x_sub = x_train[:800]
y_sub = y_train[:800]
batch_size = 64
epochs = 100
```

In [18]:
```
# as sementes ajudam a ter resultados reproduzíveis
#tf.keras.backend.clear_session()
seed(1)
set_seed(2)

Inception.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.SGD(lr=0.05, momentum=0.8),
              metrics=['accuracy'])

histInc = Inception.fit(x_sub, y_sub,
                    batch_size=batch_size,
                    epochs=epochs, verbose=0)
```

In [19]:
```
# as sementes ajudam a ter resultados reproduzíveis
#tf.keras.backend.clear_session()
seed(1)
set_seed(2)

ResNet.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.SGD(lr=0.05, momentum=0.8),
              metrics=['accuracy'])

histResNet = ResNet.fit(x_sub, y_sub,
                    batch_size=batch_size,
                    epochs=epochs, verbose=0)
```

In [20]:
```
score1T = Inception.evaluate(x_sub, y_sub, verbose = 0)
score2T = ResNet.evaluate(x_sub, y_sub, verbose = 0)

score1 = Inception.evaluate(x_test, y_test, verbose = 0)
score2 = ResNet.evaluate(x_test, y_test, verbose = 0)
```

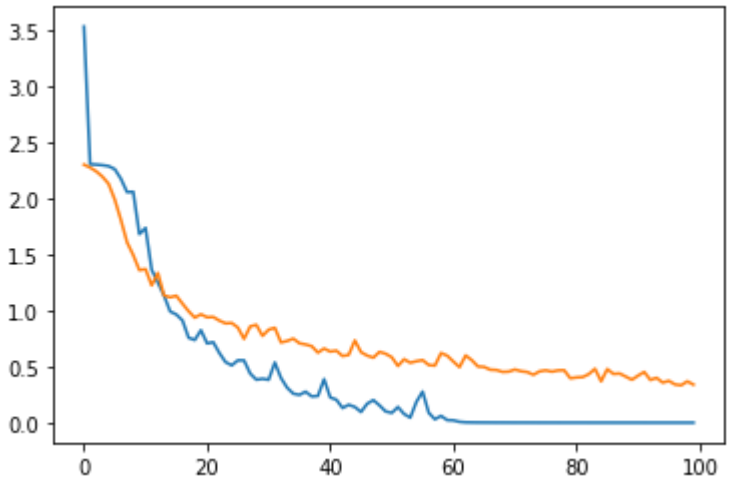
In [21]:
```
print("Inception Treinamento = Loss %.3f, Accuracy %.3f" % (score1T[
print("Inception Teste       = Loss %.3f, Accuracy %.3f" % (score1[0

print("ResNet Treinamento    = Loss %.3f, Accuracy %.3f" % (score2T[
print("ResNet Teste          = Loss %.3f, Accuracy %.3f" % (score2[0
```

```
Inception Treinamento = Loss 0.345, Accuracy 0.858
Inception Teste       = Loss 0.715, Accuracy 0.758
ResNet Treinamento    = Loss 0.000, Accuracy 1.000
ResNet Teste          = Loss 1.515, Accuracy 0.800
```

In [22]:
```
plt.plot(histResNet.history['loss'])
plt.plot(histInc.history['loss'])
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x7f4bd82f4790>]
```



In [ ]: