

MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Módulo III - Arquiteturas de CNNS e treinamento de redes profundas

Avaliação

Moacir Antonelli Ponti

CeMEAI - ICMC/USP São Carlos

As respostas devem ser dadas no Moodle, use esse notebook apenas para gerar o código necessário para obter as respostas

Questão 1)

Quais das alternativas abaixo compõe pontos fundamentais na prática para o projeto viável de treinamento de redes profundas?

- (a) Conhecer os valores possíveis da função de perda utilizada, buscando por taxas de aprendizado e batchsizes adequados ao otimizador utilizando pequenos conjuntos de treinamento e validação.
 - (b) Utilizar o maior número de instancias disponível para executar experimentos exaustivos com relação a função de custo, taxa de aprendizado, outros hiperparâmetros e escolhas como: momentum, decaimento de taxa de aprendizado, tamanho do batch, em como o otimizador utilizado.
 - (c) Utilizar os valores padrão disponíveis nos pacotes de software para aprendizado de máquina, investigando diferentes arquiteturas de rede neural populares na literatura, até encontrar aquela que gera o melhor resultado no conjunto de testes
 - (d) Inicializar sempre os pesos com valores aleatórios é suficiente para garantir a viabilidade do treinamento.
-

Questão 2)

Funções de perda diferentes geram intervalos de valores diferentes para um mesmo problema. Qual a consequência de uma função escolhida gerar valores muito

pequenos durante o treinamento?

- (a) Facilita o treinamento gerando maior generalização do modelo final
 - (b) Problemas de precisão numérica e relacionados à baixa magnitude do gradiente
 - (c) Problemas relacionados à alta magnitude do gradiente e baixa generalização do modelo
 - (d) Podem ocorrer problemas de overfitting devido à convergência do modelo acontecer muito rapidamente
-

Questão 3)

Considere um problema cuja saída desejada seja um valor contínuo entre -20 e 10. Considerando que não há possibilidade de normalizar ou transformar esse intervalo, qual par função de ativação e qual função de perda são as mais adequadas para resolver esse problema?

- (a) Ativação LeakyRelu e Perda Entropia Cruzada
 - (b) Ativação Tangente Hiperbólica e Perda Quadrática
 - (c) Ativação Relu e Perda Entropia Cruzada
 - (d) Ativação Linear e Perda pelo Erro Absoluto Médio
-

Questão 4)

Carregue a base de dados Fashion-MNIST e projete uma rede do tipo ResNetInception utilizando os módulos Inception e Residuais conforme vistos em aula. A arquitetura deve ter, nessa ordem:

- Bloco Residual com 64 filtros
- 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
- GlobalAveragePooling2D

Além disso utilizar: otimizador Adam, com batchsize 32, e 5 épocas.

Investigue os seguintes valores de taxa de aprendizado: 0.001, 0.005, 0.01 e 0.05, todos com decaimento exponencial com valor 0.05

Treine por 5 épocas, utilizando as primeiras 1000 imagens (:1000) para treinamento e as próximas 1000 para validação (1000:2000).

Qual taxa de aprendizado teve maior acurácia na validação?

- (a) 0.001

(b) 0.005

(c) 0.01

(d) 0.05

```
In [12]: import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from numpy.random import seed
from tensorflow.random import set_seed

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

fig, axes = plt.subplots(2,10, figsize=(10,2))
ax = axes.ravel()
for i in range(20):
    ax[i].imshow(x_train[i], cmap="gray")
    ax[i].axis('off')

img_lin, img_col = x_train.shape[1], x_train.shape[2]
num_classes = len(np.unique(y_train))
print(x_train.shape)
print('Classes: ', num_classes)

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

# verifica se as imagens da base de dados tem um canal (i.e. em tons de cinza)
# ou mais do que um canal e se houver mais do que um canal entao armazena a
# quantidade de canais
if (len(x_train.shape) == 3):
    n_channels = 1
else:
    n_channels = x_train.shape[3]

# re-formatando as imagens de forma que sejam transformadas em
# matrizes com canais (por exemplo quando as imagens sao RGB)
if keras.backend.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], n_channels, img_lin, img_col)
    x_test = x_test.reshape(x_test.shape[0], n_channels, img_lin, img_col)
    input_shape = (n_channels, img_lin, img_col)
else:
    x_train = x_train.reshape(x_train.shape[0], img_lin, img_col, n_channels)
    x_test = x_test.reshape(x_test.shape[0], img_lin, img_col, n_channels)
    input_shape = (img_lin, img_col, n_channels)
```

(60000, 28, 28)

Classes: 10



```
In [31]: 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, f4_out):
    # 1x1 conv
    conv1 = Conv2D(f1_out, (1,1), padding='same', activation='relu')(layer_in)
    # 3x3 conv
    conv3 = Conv2D(f2_in, (1,1), padding='same', activation='relu')(layer_in)
    conv3 = Conv2D(f2_out, (3,3), padding='same', activation='relu')(conv3)
    # 5x5 conv
    conv5 = Conv2D(f3_in, (1,1), padding='same', activation='relu')(layer_in)
    conv5 = Conv2D(f3_out, (5,5), padding='same', activation='relu')(conv5)
    # 3x3 max pooling
    pool = MaxPooling2D((3,3), strides=(1,1), padding='same')(layer_in)
    pool = Conv2D(f4_out, (1,1), padding='same', activation='relu')(pool)
    layer_out = concatenate([conv1, conv3, conv5, pool], axis=-1)
    return layer_out

def residual_block(layer_in, n_filters):
    merge_input = layer_in
    #verifica se é necessária uma primeira camada para deixar o número de filtros
    if layer_in.shape[-1] != n_filters:
        merge_input = Conv2D(n_filters, (1,1), padding='same', activation='relu',
    # conv1
    conv1 = Conv2D(n_filters, (3,3), padding='same', activation='relu', kernel_in
    # conv2
    conv2 = Conv2D(n_filters, (3,3), padding='same', activation='linear', kernel_i
    # 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
```

In [32]:

Questão 5)

Considere as situações abaixo listadas, observadas de forma independente após treinar modelos de rede profunda por 50 épocas para um problema de classificação com 20 classes usando a perda quadrática:

- I - O valor da perda convergiu para um valor próximo a zero
- II - O valor da perda após a primeira época foi de 2.61
- III - A acurácia final medida no treinamento foi de 65% e na validação de 68%
- IV - O valor da perda caiu de 0.90 na primeira época para 0.72 na última época

OBS: para interpretar os valores acima, calcule a perda quadrática e a acurácia para os seguintes cenários de vetores de probabilidade de saída (para 20 classes): todas as classes equiprováveis (aleatório uniformemente distribuído), e de um vetor de saída com a classe correta, porém com baixa probabilidade, com 0.2 para a classe correta, mas que possui outros 8 valores com 0.1.

Quais das situações acima estão tipicamente ligadas a problemas no cálculo do gradiente (causado muitas vezes por escolhas erradas na arquitetura e função e custo) e/ou na convergência do modelo causados por escolhas erradas do otimizador e seus hiper-parâmetros?

- (a) Todos
- (b) II e IV
- (c) I e II
- (d) III e IV

In []: