MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Módulo VIII - Tópicos em Deep Learning

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)

Leia sobre o dataset e desafio COCO Keypoints em https://cocodataset.org/#keypoints-2020, no qual o objetivo é detectar pessoas numa cena e localizar seus pontos chave.

Qual abordagem, dentre as abaixo, seria adequada para esse cenário?

(a) Rede neural convolucional treinada para detectar a presença de uma ou mais pessoas, bem como realizar regressão dos pontos chave de cada pessoa por meio do projeto de uma camada de saída com essas informações
(b) Rede neural convolucional para classificar entre imagens que possuem pessoas e imagens que não possuem pessoas, aprendendo a detecção de pessoas por meio de uma função de entropia cruzada a qual deverá computar a probabilidade de uma ou mais pessoas estarem na cena.
(c) Rede neural convolucional com função de perda triplet aproximar pontos chave de uma pessoa independente de sua pose
(d) Rede neural densa com emprego de uma função de custo que combine classificação e intersecção sobre união para gerar como saída a sobreposição entre áreas ocupadas por pessoas preditas e reais

Justificativa: Dentre as opções, a mais adequada é uma rede neural cuja saída contenha valores previstos para pontos chave e que também detectem a presença de pessoas para guiar o treinamento.

Exercício 2)

Assuma um cenário em que temos uma grande base de dados com documentos, em formato texto, escrito por 200 autores de uma editora. Para fins de detecção de plágio queremos aprender uma representação que seja capaz de projetar um texto num espaço de

características (espaço latente) no qual seja possível medir a similaridade entre esse texto de entrada e um texto da base de dados, identificando de qual autor possui maior similaridade. Como principal problema, o texto possui alta sobreposição de conteúdo com trechos em comum entre autores arbitrários, sendo que os trechos que caracterizam cada autor representa parte significativamente menor do texto, o que é difícil de detectar e pré-processar (impedindo por exemplo remover esses trechos de confusão).

Tendo em mãos um word-embedding adequado, qual seria a melhor opção para aprender uma representação útil, considerando as abordagens:

I - Camada GRU
II - Camada densa de projeção em k dimensões
III - Camada densa de classificação com 200 neurônios
IV - Função entropia cruzada
V - Função contrastiva

(a) Treinar rede com camadas I, II, III e função V
(a) Treinar rede com camada II e função IV
(c) Treinar rede com camadas I, II e III com função IV, e depois novamente as camadas I e II e a função V, retirando a IV
(d) Treinar rede com camadas I e III com função V, e depois acrescentar a camadas II treinando com a função IV

Justificativa: O uso de GRU pode ser importante para capturar estruturas sequenciais no texto; a camada densa de projeção também é relevante para aprendermos um espaço de características com k dimensões. No entanto, a classificação unicamente (II + IV) pode ser insuficiente num cenário em que há alta sobreposição de características. Assim, seria uma melhor opção começar aprendendo um classificador, mas posteriormente treinar as camadas I e II com a perda contrastiva (V).

Exercício 3)

Para qual cenário a auto-supervisão é uma técnica útil?

(a) Processar dados não estruturados de qualquer origem criando um classificador diretamente a partir dos dados sem a necessidade de rotulação, e posteriormente classificar novas imagens utilizando esse modelo.
(b) Pré-processar grandes bases de dados de forma a eliminar outliers, ruído e outros efeitos indesejados nessa base de dados, garantindo maior acurácia para dados futuros.
(c) Coletar e utilizar grandes quantidades de dados não anotados, e obter um modelo inicial que possa ser utilizado como base para adaptar modelos para problemas específicos, vencendo a barreira de anotar dados massivamente.
(d) Para base de dados pequenas em que não se dispõe de grande quantidades de exemplos para treinar redes neurais profundas para tarefas de classificação e regressão, pois essa técnica realiza aumento de dados de forma nativa.

Justificativa: Auto-supervisão tem a intenção de diminuir a dependência de grandes

quantidades de dados rotulados, permitindo que se utilize dados raspados da Internet ou obtidos de forma automática para gerar um modelo inicial que possa ser transferido para tarefas específicas.

Exercício 4)

Vamos implementar as funções contrastive e triplet para observar se essas funções são minimizadas após o treinamento de uma CNN com a camada softmax e entropia cruzada. Em resumo, não iremos treinar a rede com as funções contrastive/triplet, mas apenas avaliar o espaço de características.

Use as implementações abaixo. Elas foram feitas para maior entendimento e não para velocidade. Caso necessário, o ponto principal de melhoria é a função `np.where()` que poderia ser substituída por algo mais eficiente.

Carregue a base de dados Fashion MNIST e monte uma CNN com a seguinte arquitetura:

- Convolucional com 32 filtros 3x3 e strides (2,2), sem zero pading, ativação relu
- Convolucional com 64 filtros 3x3 e strides (2,2), sem zero pading, ativação relu
- Convolucional com 64 filtros 3x3 e strides (2,2), sem zero pading, ativação relu
- Densa com 64 neurônios e ativação relu (embedding)
- Dropout 0.25
- Densa softmax com 10 neurônios

Defina as sementes seed(1), set_seed(2) e compile o modelo com o otimizador Adam, taxa de aprendizado 0.001. Após isso, monte um modelo que gere como saída a camada Densa de 64 neurônios (embedding). Obtenha as representações do embedding (antes do treinamento) para os 50 primeiros exemplos de treinamento. Depois:

1. Defina random.seed(1)
2. Compute e armazene as funções contrastive e triplet nesses 50 exemplos: note que é preciso passar as classes (em formato numérico) e as representações para a função.

Depois, com 20 épocas e batchsize = 32, treine com o otimizador Adam, taxa de aprendizado 0.001, e usando apenas as 10000 primeiras instâncias da base [:10000].

Obtenha as representações do embedding após o treinamento para os 50 primeiros exemplos de treinamento. A seguir:

1. Defina random.seed(1)
2. Compute e armazene as funções contrastive e triplet nas representações dos mesmos 50 exemplos usando o modelo treinado

Considerando os valores das perdas arredondados para 4 casas decimais e o efeito do treinamento nas perdas contrastive e triplet, qual foi o resultado?

OBS: para que essas funções possam ser usadas para treinamento no Keras é preciso utilizar funções do tensorflow ao invés de numpy, além de garantir que o `shape` de `y_true` é igual a de `y_pred`. Caso necessário, adapte e procure como passar essa

função de perda personalizada.

(a) Valor da triplet é maior do que o da contrastiva, e apenas a triplet reduziu seu valor significativamente, a contrastive teve redução na 4.a casa decimal
(b) Valor da triplet é menor do que o da contrastiva, e ambas foram minimizadas significativamente (na ordem da 2.a casa decimal ou maior) após o treinamento
(c) Valor da triplet é maior do que o da contrastiva, e ambas foram minimizadas significativamente (na ordem da 2.a casa decimal ou maior) após o treinamento
(d) Valor da triplet é menor do que o da contrastiva, e apenas a contrastive reduziu seu valor significativamente, a triplet teve redução na 4.a casa decimal

Justificativa: ver código abaixo

In [1]:
```
def triplet_loss(y_true, y_pred):
    """ Triplet loss
        y_true: classes dos exemplos de forma numérica (não one-hot-encodir
        y_pred: predicao da rede em termos das *representacoes*/embedding a
        Moacir A. Ponti/2020
    """
    L = 0 # loss acumulada
    m = 1 # margem
    size = y_true.shape[0] # tamanho do conjunto a avaliar

    # normalizar dados 0-1
    y_pred = (y_pred-np.min(y_pred))/(np.max(y_pred)-np.min(y_pred))

    # seleciona uma ancora por exemplo no conjunto
    for i in range(size):
        a = y_pred[i] # representacao da ancora
        y = y_true[i] # classe da ancora

        # seleciona exemplo aleatorio, positivo e negativo
        pos_ind = np.where(y_true == y)
        neg_ind = np.where(y_true != y)
        p = y_pred[pos_ind[0][random.randint(0,len(pos_ind[0])-1)]]
        n = y_pred[neg_ind[0][random.randint(0,len(neg_ind[0])-1)]]

        # computa triplet loss
        lapn = np.max([0, m + np.sum(np.power(a-p,2)) - np.sum(np.power(a-
        # acumula
        L += lapn

    return L/float(size)
```

In [2]:
```
def contrastive_loss(y_true, y_pred):
    """ Contrastive loss
        y_true: classes dos exemplos de forma numérica (não one-hot-encodir
        y_pred: predicao da rede em termos das *representacoes*/embedding a
        Moacir A. Ponti/2020
    """
    L = 0
    m = 1
    size = y_true.shape[0]

    # normalizar dados
    y_pred = (y_pred-np.min(y_pred))/(np.max(y_pred)-np.min(y_pred))

    # para cada exemplo "ancora"
    for i in range(size):
        a = y_pred[i] # representacao a
        y_a = y_true[i] # classe de a

        # sorteia exemplo (nao ancora)
        ind = i
        while (ind == i):
            ind = random.randint(0,size-1)

        p = y_pred[ind] # representacao p

        # calcula contrastive loss (com base na classe)
        if (y_true[ind] == y_a):
            lapn = np.sum(np.power(a-p,2))/2.0
        else:
            lapn = np.max([0, m - np.sum(np.power(a-p, 2)) ])/2.0

        L += lapn

    return L/float(size)
```

In [3]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from numpy.random import seed
from tensorflow.random import set_seed
from tensorflow import keras
from tensorflow.keras import layers
```

In [4]:
```
# 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()
```

In [5]:
```
# obtendo informações das imagens (resolucao) e dos rótulos (número de cla
img_lin, img_col = x_train.shape[1], x_train.shape[2]
if (len(x_train.shape) == 3):
      n_channels = 1
else:
      n_channels = x_train.shape[3]

num_classes = len(np.unique(y_train))
print(x_train.shape)
print('Classes: ', num_classes)

# 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_one_hot = keras.utils.to_categorical(y_train, num_classes)
y_test_one_hot = keras.utils.to_categorical(y_test, num_classes)

# formatar para treinamento
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)

# formato da entrada
input_shape = (img_lin, img_col, n_channels)
print('Tamanho da entrada: ', input_shape)
```

```
(60000, 28, 28)
Classes:  10
Tamanho da entrada:  (28, 28, 1)
```

In [6]:
```
def cnn(input_shape, n_classes, verbose=False):
    CNN = keras.Sequential()
    CNN.add(keras.layers.Conv2D(32, kernel_size=(3,3), strides=(2,2), padd
    CNN.add(keras.layers.Conv2D(64, kernel_size=(3,3), strides=(2,2), padd
    CNN.add(keras.layers.Conv2D(64, kernel_size=(3,3), strides=(2,2), padd
    CNN.add(keras.layers.Flatten())
    CNN.add(keras.layers.Dense(64, activation='relu', name='embedding'))
    CNN.add(keras.layers.Dropout(0.25))
    CNN.add(keras.layers.Dense(n_classes, activation='softmax'))
    if verbose: CNN.summary()
    return CNN
```

In [7]:
```
epochs = 20
batch_size = 32
x_subset = x_train[:10000]
y_subset = y_train_one_hot[:10000]
```

In [8]:
```
# as sementes ajudam a ter resultados reproduzíveis
seed(1)
set_seed(2)

cnet1 = cnn(input_shape, num_classes, verbose=True)
cnet1.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 13, 13, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 6, 6, 64)          18496
```

```
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 2, 2, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
embedding (Dense)            (None, 64)                16448
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 10)                650
=================================================================
Total params: 72,842
Trainable params: 72,842
Non-trainable params: 0
```

In [9]:
```
embedding_layer = keras.models.Model(inputs=cnet1.input, outputs=cnet1.get
```

In [10]:
```
emb_train = embedding_layer.predict(x_train)
```

In [25]:
```
random.seed(1)
cont_random = contrastive_loss(y_train[:50], emb_train[:50])
trip_random = triplet_loss(y_train[:50], emb_train[:50])
```

In [13]:
```
cnet1.fit(x_subset, y_subset, batch_size=batch_size, epochs=epochs, verbos
```

Out[13]:
```
<tensorflow.python.keras.callbacks.History at 0x7f943240fa30>
```

In [14]:
```
emb_train2 = embedding_layer.predict(x_train)
```

In [26]:
```
random.seed(1)
cont_train = contrastive_loss(y_train[:50], emb_train2[:50])
trip_train = triplet_loss(y_train[:50], emb_train2[:50])
```

In [28]:
```
print("Contrastive: antes %.4f, depois %.4f" % (cont_random, cont_train))
print("Triplet: antes %.4f, depois %.4f" % (trip_random, trip_train))
```

```
Contrastive: antes 0.1603, depois 0.0673
Triplet: antes 0.3183, depois 0.1004
```

Exercício 5)

Suponha uma base de dados que possua 32 características relacionadas a transações financeiras realizadas por clientes de um banco. Temos disponível 10 milhões de transações não anotadas de diferentes clientes, e transações anotadas para 20 mil clientes: em número girando entre 100 e 2 mil transações anotadas em "normais", "suspeitas" e "anomalias" por cliente.

Desejamos criar modelos de detecção de anomalias em transações para clientes recentes do banco, para os quais temos nenhum ou poucos dados de transações. Para evitar que as primeiras predições sejam aleatórias, considere as seguintes abordagens para aprender uma representação com 32 características que possa ser usada como modelo inicial para posterior ajuste com dados futuros rotulados:

I - denoising autoencoder overcomplete com restrição de esparsidade no código de 32

dimensões
II - rede neural que receba por entrada as 32 dimensões embaralhadas e que aprenda a detectar as posições corretas das características
III - rede neural treinada com transações anotadas de clientes com perfil semelhante ao novo cliente

Essas abordagens dizem respeito a:

(a) I - auto-supervisão, II - análise de permutações, III - aprendizado ativo
(b) I - aprendizado não-supervisionado, II e III - auto-supervisão
(c) I - redução de dimensionalidade, II - auto-supervisão, III - transferência de aprendizado
(d) I e II - auto-supervisão, III - transferência de aprendizado

Justificativa: Auto-supervisão é qualquer tarefa cuja função de custo possa ser computada a partir dos dados de entrada e que não dependa de anotação. Transferência de aprendizado é o processo de usar um modelo treinado em outra tarefa. Na abordagem III temos uma rede neural treinada com anotação e portanto não é auto-supervisionada. Assumimos que o modelo de III é tranferível para a tarefa de um novo cliente. As abordagens I e II são auto-supervisionadas pois podemos treinar redes neurais com as 10 milhões de transações disponíveis.

In [ ]: