MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Módulo V - Redes neurais auto-associativas e geradoras

Exercícios

Moacir Antonelli Ponti

CeMEAI - ICMC/USP São Carlos

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

Exercício 1)

Auto-encoders são apropriados para tarefas de aprendizado não supervisionado, em que os dados de entrada são reconstruídos e uma representação compacta destes dados é aprendida. Em relação a esse método, qual afirmação está incorreta?

a) A função de custo Mean Square Error deve ser utilizada com o propósito de medir a eficiência do aprendizado da rede, em que se considera seu potencial de obter imagens com alto grau de similaridade em relação às imagens de entrada.

b) Podemos utilizar um auto-encoder para ser treinado com exemplos não rotulados e reaproveitar somente o Encoder para formar uma CNN com adição de novas camadas para a predição com posterior fine-tuning.

c) Auto-encoders overcomplete obtém naturalmente códigos compactos e de baixa dimensionalidade, similar a projeção PCA nos principais componentes considerando dimensionalidade menor do que a da entrada.

d) Denoising auto-encoderes são arquiteturas que podem ser utilizadas para duas tarefas simultaneamente após o seu treinamento. O primeiro deles é fornecer espaço de características que representam um conjunto de dados. O segundo propósito é funcionar como um filtro para remoção de ruídos.

Exercício 2)

É correto afirmar sobre a principal tarefa realizada pela família de métodos do tipo Autoencoder

(a) Mapeia os dados de entrada em um espaço latente que melhor permita uma boa classificação dos dados de treinamento, sendo o foco principal do aprendizado aumento de

acurácia
(b) Mapeia os dados de entrada em um espaço latente, e posteriormente aprende uma reconstrução desse espaço latente novamente no espaço de entrada, sendo o foco principal do aprendizado obter um espaço latente compacto e representativo.
(c) Mapeia os dados de entrada no espaço de saída num problema similar à regressão, em que temos informações a priori sobre como reconstruir os dados de entrada
(d) Mapeia os dados de entrada com principal objetivo de eliminar o ruído dos dados e portanto é útil para tarefas de limpeza de dados.

Exercício 3)

Considere os métodos Autoencoder, Denoising Autoencoder e Variational Autoencoder. Podemos dizer que esses métodos se enquadram em qual tipo de aprendizado?

(a) Supervisionado
(b) Semi-supervisionado
(c) Não supervisionado
(d) Fracamente supervisionado

Exercício 4)

Você treinou uma GAN, porém enquanto a perda do discriminador esteve sempre decrescente a perda do gerador cresceu no treino. Qual dos procedimentos abaixo tem a **menor** chance de auxiliar no treinamento:

a) Reduzir a taxa de aprendizado do discriminador
b) Aumentar a capacidade do discriminador e reduzir a do gerador
c) Regularizar ou aumentar a regularização do discriminador
d) Fazer mais de uma atualização do gerador para cada atualização dos pesos do discriminador

Exercício 5)

Em geral qual modelo: Variational Autoencoder (VAE) ou Generative Adversarial Network (GAN) é mais difícil de treinar e por que?

a) GANs, pois estamos tentando otimizar 2 modelos simultâneamente e de forma adversarial
b) VAEs, pois sua perda é complexa de calcular
c) GANs, pois necessitam de muita aumentação de dados
d) VAEs, pois precisam aprender distribuições de dados para dois modelos (encoder e decoder)

Exercício 6)

Carregue a base de dados `smartphone_activity_dataset.csv` , que possui 6

classes, conforme abaixo. Utilizaremos os primeiros 70% exemplos como treinamento e o restante como teste.

Defina as sementes `seed(1)` e `set_seed(2)`. Logo após, projete e instancie um Encoder com as seguintes camadas, sendo que, após a entrada, todas terão ativação tangente hiperbólica:

- Entrada
- Densa com metade das dimensões da entrada (use a divisão inteira //2)
- Densa com um quarto das dimensões da entrada (use a divisão inteira //4)
- Densa com 32 dimensões

Sem realizar treinamento, passe os dados de treinamento e teste pela rede, obtendo as ativações da última camada, com 32 dimensões.

Carregue o classificador SVC da biblioteca sklearn, e treine com parâmetros C=1, random_state=1, com as características obtidas da rede neural de treinamento, realizando a predição no teste a seguir, medindo a acurácia por meio da função score.

Qual foi o resultado de acurácia obtido?

(a) Acurácia de um classificador aleatório
(b) Acurácia abaixo de 5%
(c) Acurácia no intervalo entre 45% e 65%
(d) Acurácia acima de 75%

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

In [ ]:
```
df = pd.read_csv("smartphone_activity_dataset.csv")
df.head()
```

In [ ]:
```
rotulos = np.array(df['activity'])
features = np.array(df.iloc[:, :-1])

print(features.shape)
perc_train = 0.7

n_train = int(features.shape[0]*perc_train)
n_test = int(features.shape[0]*(1-perc_train))
print(n_train)
print(n_test)

x_train = features[:n_train,:]
y_train = rotulos[:n_train]

x_test = features[n_train:,:]
y_test = rotulos[n_train:]
```

In [ ]:
```
### autoencoder
### projecao sem treinamento
### treinamento SVM
```

Exercício 7)

Uso de GANs para aprendizado de representações. Utilizando a mesma base de dados do exercício anterior, projete e treine uma GAN para gerar exemplos artificiais da base de dados `smartphone_activity_dataset.csv` . Utilizaremos todos os exemplos para treinamento da GAN.

Para isso, utilize a GAN proposta em aula conforme o código abaixo, alterada conforme o código abaixo e as seguintes configurações:

- discriminador com uma camada oculta com 32 dimensões (utilize o parâmetro `name='embedding'` nessa camada para facilitar seu uso posterior), deve ser treinado com Adam e taxa de aprendizado 0.0001 (apenas para o discriminador)
- gerador com 2 camadas ocultas com 256 neurônios
- dimensão de z = 256
- dimensão de entrada igual a da base de dados
- função de distribuição alvo deve amostrar da base de dados

Todas as camadas serão `tanh` exceto a camada de saída do discriminador que é `sigmoid` .

Treine a GAN por 1000 épocas com batch size 32 e otimizador Adam com lr=0.001.

Após, utilize a camada "embedding" do discriminador como extratora de características, passando os dados da base de treinamento e teste (na mesma divisão feita na questão anterior) para o discriminador, e pegando a projeção feita pela camada densa.

Treine classificadores SVM com parâmetros C=1 e random_state=1, um na base de dados com suas características originais, e o outro nas característias do "embedding" do discriminador. Obtenha o score nos respectivos conjuntos de teste. Os resultados estão em qual intervalo?

(a) Original = [92,97], Embedding 32d GAN = [88,92], a GAN foi capaz de aprender uma boa representação compacta com acurácia próxima da original sendo a quantidade de épocas suficiente para esse aprendizado
(b) Original = [85,88], Embedding 32d GAN = [85,88], a GAN produz representação similar aos dados originais, porém com menor dimensionalidade, indicando que houve convergência
(c) Original = [92,97], Embedding 32d GAN = [10,20], a GAN gera uma representação que produz classificação próxima da aleatória
(d) Original = [92,97], Embedding 32d GAN = [79,82], a GAN gera uma representação compacta com resultado abaixo da original, sendo necessário executar por mais épocas para investigar se é capaz de gerar uma representação mais fiel aos dados

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

In [ ]:
```
## funcoes discriminador, gerador e amostragem de exemplos falsos/verdadei

# modelo discriminador base : classificador
def discriminator(dim_input=2, embedding_size=128):
    model = models.Sequential()
    model.add(layers.Dense(embedding_size, activation='tanh', name='embedd
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=keras.optimizers.A
    return model

def distribuicao_alvo(n, x_train):
    labels_reais = np.ones((n,1))
    return x_train[np.random.choice(x_train.shape[0], n, replace=False), :

# modelo gerador, cuja entrada tem dimensão do espaço latente
# sua saída tem dimensão igual a dos exemplos da distribuição alvo
def generator(z_dim, dim_output=2, embedding_size1=256, embedding_size2=25
    model = models.Sequential()
    model.add(layers.Dense(embedding_size1, activation='tanh', name='embed
    model.add(layers.Dense(embedding_size2, activation='tanh', name='embed
    model.add(layers.Dense(dim_output, activation='tanh'))
    return model

# funcao para gerar uma amostra com n exemplos da distribuicao latente
def amostra_distribuicao_latente(z_dim, n):
    exemplos_z = np.random.randn(z_dim * n)
    return exemplos_z.reshape(n, z_dim)

# funcao para gerar exemplos "falsos", aleatorios a partir da
# saída do gerador G(z)
def gera_exemplos_falsos(model_g, z_dim, n):
    exemplos_z = amostra_distribuicao_latente(z_dim, n)
    exemplos_falsos = model_g.predict(exemplos_z)
    labels_falsos = np.zeros((n,1))
    return exemplos_falsos, labels_falsos
```

In [ ]:
```
## funcao para o modelo e treinamento da GAN

## instanciamento do discriminador, gerador e GAN

## treinamento GAN

## uso do embedding GAN para treinar SVM
```

Exercício 8)

Interpolação utilizando código autoencoder.

Crie um Autoencoder profundo do tipo Denoising Undercomplete para imagens da base de dados MNIST.

Para tornar o treinamento mais rápido utilizaremos apenas as 3000 primeiras imagens da base de dados de treinamento. Crie uma versão ruidosa do conjunto de treinamento conforme indicado no código.

O Autoencoder deve possuir a seguinte arquitetura no encoder (todas as camadas com ativação relu):

- Conv2D com 32 filtros 3x3, strides 2, zeropadding
- Conv2D com 32 filtros 3x3, strides 2, sem zeropadding
- MaxPooling2D com poolsize=2
- Densa com 32 dimensões (utilize name='code' para facilitar)

A seguir, deve possuir um decoder espelhado, resultando na saida de mesma dimensão da entrada.

Antes de instanciar o modelo, inicialize as sementes com seed(1), set_seed(2). Depois, compile e treine com perda MSE, Otimizador SGD, taxa 0.01, momentum 0.95, por 25 épocas com batchsize 32.

Vamos agora obter interpolações de imagens utilizando os seus códigos. Para isso crie dois modelos, com base no treinado:

1. Encoder, que permite recuperar o código de uma imagem (camada 'code')
2. Decoder, que permite receber um código e retornar uma imagem

O segundo é mais complexo, exige que seja feita a montagem das camadas utilizando a seguinte instrução (estude com calma para entender):

In [ ]:
```
## cria nova camada de entrada para receber dimensionalidade do código
#input_code_layer = keras.layers.Input(shape=(code_dim))

## encadeia camadas a partir de 'inicio', nesse caso 6 - ajuste conforme ne
#inicio = 6
#x = input_code_layer
#for layer in convae.layers[inicio:]:
#    x = layer(x)

#decoder = keras.models.Model(inputs=input_code_layer, outputs=x)
```

Obtenha as imagens de teste de índice 128 e 198, e então:

1. calcule seus códigos pelo encoder: code128 e code198;
2. gere interpolações 10 lineares entre code128 e code198 combinando linearmente cada dimensão do vetor com ponderações entre 0 (que resulta apenas na imagem 128) e 1 (que resulta apenas na imagem 198);
3. passe cada vetor interpolado pelo decoder, obtendo imagens intermediárias;

OBS: os modelos esperam arrays em formatos específicos para predição, para passar uma única imagem use `model.predict(np.array([x_test[indice_imagem],]))`

Quais das imagens abaixo foi obtida?

(a)

(b)

(c)

(d)

In [ ]:
```
from tensorflow import keras
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# os pixels das imagens sao reescalados para melhor processamento
# em particular divide-se por 255 para que os valores fiquem entre 0 e 1
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
print('Dataset size:')
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

img_rows, img_cols = x_train.shape[1], x_train.shape[2]
input_shape = (img_rows, img_cols, 1, )
n_classes = 10

n=10
plt.figure(figsize=(10, 5))
for i in range(n):
    ax = plt.subplot(2, n, i+1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()

seed(1)
set_seed(2)
noisy_train = x_train[:3000] + np.random.random(x_train[:3000].shape)*0.3

n=10
plt.figure(figsize=(10, 5))
for i in range(n):
    ax = plt.subplot(2, n, i+1)
    plt.imshow(noisy_train[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```

In [ ]:
```
## projeta e treina autoencoder
## monta modelo que aproveita apenas encoder
## monta modelo que aproveita apenas decoder
## obtem encoding para as imagens
## processa encodings por interpolacao, passando para o decoder
## visualiza imagens reconstruídas
```