عنوان : تمرین سوم یادگیری عمیق

نگارنده : سحر داستانی اوغانی

شماره دانشجویی : ۹۹۱۱۲۱۰۸

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

بخش صفر: Autoencoder

معرفی

اتوانکدرها، نوع خاصی از معماری شبکه‌های عصبی هستند که ورودی، خروجی آن‌ها مشابه یکدیگر است. آن‌ها به روش غیر نظارتی آموزش دیدند و وظیفه‌ی رگرسیون داده‌ها را بر عهده دارند. درواقع از یک شبکه‌ی اتوانکدر برای پیش‌بینی ورودی مدل استفاده می‌شود. این شبکه‌ها خود را وادار می‌کنند که نمایشی موثر از ورودی را به شیوه‌ای ارائه دهند که **bottleneck**، فشرده شده‌ی ورودی در ابعاد پایین باشد، تا بتوان توسط یک **decoder**، آن را به ورودی تبدیل کرد و خروجی را از آن دریافت کرد.

معماری

این شبکه‌های از سه لایه‌ی زیر تشکیل شدند. عملکرد هر لایه را به صورت جداگانه در ادامه توضیح خواهیم داد.

- لایه Encoding

این لایه، شامل سری‌ای از لایه‌هاست که تعداد نودها در آن‌ها به حدی رو به کاهش است که در نهایت به نمایش نهایی لایه‌ی پنهان کاهش می‌یابد.

- لایه نمایش پنهان

این لایه، فضای رتبه پایینی را نمایش می‌دهد که در آن، ورودی‌ها کاهش یافته و اطلاعات آن‌ها حفظ شده است.

- لایه‌ی Decoding

این لایه تصویر آینه‌ای لایه‌ی Encoding است ولی تعداد ندها در هر لایه افزایش می‌یابد تا در نهایت خروجی مشابه ورودی را تولید کند.

کاربرد

- کاهش بعد
- فشرده‌سازی تصویر
- تخلیه تصویر از نویز
- تولید تصویر
- استخراج ویژگی

بخش اول: Image Reconstruction

دیتاست و پیش‌پردازش

در این بخش می‌خواهیم اتوانکدری طراحی کنیم که بتواند تصاویر دیتاست Fashion Mnist را بازسازی کند. برای اینکار از keras کمک می‌گیریم. بنابراین در ابتدا کتابخانه‌های لازم را فراخوانی می‌کنیم.

```
[1]  1 import numpy as np
     2 import pandas as pd
     3 from random import randint
     4 from imgaug import augmenters
     5 from keras.models import Model
     6 import matplotlib.pyplot as plt
     7 from numpy import argmax, array_equal
     8 from keras.utils import to_categorical
     9 from keras.datasets import fashion_mnist
    10 from keras.callbacks import EarlyStopping
    11 from sklearn.model_selection import train_test_split
    12 from keras.layers import Dense, Input, Conv2D, LSTM, MaxPool2D, UpSampling2D
```

سپس دیتاست را فراخوانی کرده و آن را پیش‌پردازش می‌کنیم. لازم به ذکر است که برای فراخوانی دیتاست در این تمرین از کتابخانه‌ی آنلاین Fashion Mnist در leras.dataset استفاده شده است. شکل ظاهری داده‌های این دیتاست قبل از انجام پیش‌پردازش به صورت زیر می‌باشد:

```
1 print('train_x size :',train_x.shape)
2 print('train_y size :',train_y.shape)
3
4 print('test_x size :',test_x.shape)
5 print('test_y size :',test_y.shape)
```

```
train_x size : (60000, 28, 28)
train_y size : (60000,)
test_x size : (10000, 28, 28)
test_y size : (10000,)
```

پس از انجام عملیاتی مانند reshape کردن، تبدیل آن از integer به float و نرمالایز کردن آن‌ها به بازه‌ی ۰ تا ۱، فرمت داده‌ها تغییر می‌کنند.

```
--------------------Dataset is reshaped
--------------------Dataset is converted to float
--------------------Dataset is normalized
```

برای اینکه بتوان فرایند پیش‌بینی را بر روی داده‌ها(در آینده) انجام داد، لازم است دیتاست را به دو گروه آموزش و ولیدیشن تقسیم بندی کنیم. در این صورت، شکل ظاهری آن‌ها به فرمت زیر در می‌آید:

```
train_x size : (48000, 784)
val_x size : (12000, 784)
```

طراحی معماری اتوانکدر

همانطور که در صورت سوال گفته شده است، لازم است شبکه را به نحوی طراحی کنیم که encoder، از سه لایه با تعداد نودهای ۲۰۰۰، ۱۲۰۰، ۵۰۰ تشکیل شده باشد و decoder مانند آینه عمل کند یعنی دارای لایه‌های ۵۰۰، ۱۲۰۰، ۲۰۰۰ باشد. بنابراین کافی است شبکه را به صورت زیر طراحی کنیم. لازم به ذکر است، لایه‌ی پنهان را با ۱۰ نود فرض کردیم، زیرا پس از آزمون و خطا متوجه می‌شویم که همین ۱۰ نود می‌تواند به خوبی فشرده شده‌ی ورودی را به خروجی تبدیل کند.

```
#input layer
input_layer = Input(shape=(784,))

#encoding layer
encode_layer1 = Dense(2000, activation='relu')(input_layer)
encode_layer2 = Dense(1200, activation='relu')(encode_layer1)
encode_layer3 = Dense(500, activation='relu')(encode_layer2)

## latent view
latent_view   = Dense(10, activation='sigmoid')(encode_layer3)

#decoding layer
decode_layer1 = Dense(500, activation='relu')(latent_view)
decode_layer2 = Dense(1200, activation='relu')(decode_layer1)
decode_layer3 = Dense(2000, activation='relu')(decode_layer2)

#output layer
output_layer  = Dense(784)(decode_layer3)

model = Model(input_layer, output_layer)
model.summary()
```

مدل طراحی شده دارای summary زیر می‌باشد:

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 784)]             0
_________________________________________________________________
dense (Dense)                (None, 2000)              1570000
_________________________________________________________________
dense_1 (Dense)              (None, 1200)              2401200
_________________________________________________________________
dense_2 (Dense)              (None, 500)               600500
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5010
_________________________________________________________________
dense_4 (Dense)              (None, 500)               5500
_________________________________________________________________
dense_5 (Dense)              (None, 1200)              601200
_________________________________________________________________
dense_6 (Dense)              (None, 2000)              2402000
_________________________________________________________________
dense_7 (Dense)              (None, 784)               1568784
=================================================================
Total params: 9,154,194
Trainable params: 9,154,194
Non-trainable params: 0
_________________________________________________________________
```

حال باید مدل ساخته شده را بر روی داده‌های آموزشی، آموزش دهیم. اینکار را با استفاده از early stopping انجام می‌دهیم.

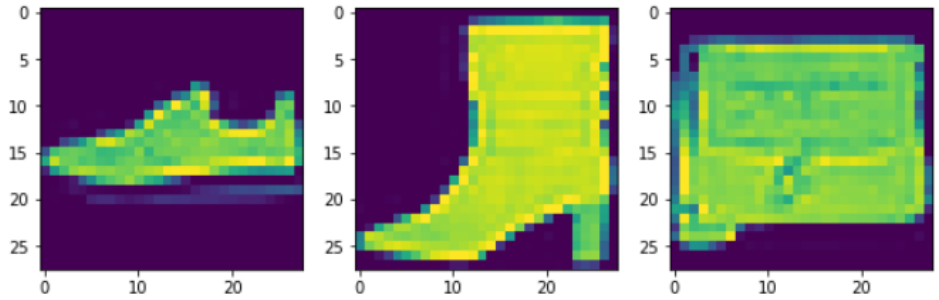
نتایج آن پس از ۲۰ epoch به شرح زیر است:

```
Epoch 1/20
24/24 [==============================] - 49s 2s/step - loss: 0.1186 - val_loss: 0.0670
Epoch 2/20
24/24 [==============================] - 48s 2s/step - loss: 0.0629 - val_loss: 0.0479
Epoch 3/20
24/24 [==============================] - 49s 2s/step - loss: 0.0455 - val_loss: 0.0396
Epoch 4/20
24/24 [==============================] - 48s 2s/step - loss: 0.0390 - val_loss: 0.0370
Epoch 5/20
24/24 [==============================] - 48s 2s/step - loss: 0.0361 - val_loss: 0.0334
Epoch 6/20
24/24 [==============================] - 48s 2s/step - loss: 0.0327 - val_loss: 0.0306
Epoch 7/20
24/24 [==============================] - 49s 2s/step - loss: 0.0297 - val_loss: 0.0279
Epoch 8/20
24/24 [==============================] - 49s 2s/step - loss: 0.0272 - val_loss: 0.0260
Epoch 9/20
24/24 [==============================] - 48s 2s/step - loss: 0.0257 - val_loss: 0.0247
Epoch 10/20
24/24 [==============================] - 48s 2s/step - loss: 0.0245 - val_loss: 0.0237
Epoch 11/20
24/24 [==============================] - 48s 2s/step - loss: 0.0236 - val_loss: 0.0230
Epoch 12/20
24/24 [==============================] - 46s 2s/step - loss: 0.0227 - val_loss: 0.0223
Epoch 13/20
24/24 [==============================] - 46s 2s/step - loss: 0.0220 - val_loss: 0.0222
Epoch 14/20
24/24 [==============================] - 48s 2s/step - loss: 0.0217 - val_loss: 0.0212
Epoch 15/20
24/24 [==============================] - 48s 2s/step - loss: 0.0208 - val_loss: 0.0206
Epoch 16/20
24/24 [==============================] - 47s 2s/step - loss: 0.0211 - val_loss: 0.0203
Epoch 17/20
24/24 [==============================] - 47s 2s/step - loss: 0.0200 - val_loss: 0.0200
Epoch 18/20
24/24 [==============================] - 47s 2s/step - loss: 0.0197 - val_loss: 0.0205
Epoch 19/20
24/24 [==============================] - 46s 2s/step - loss: 0.0196 - val_loss: 0.0192
Epoch 20/20
24/24 [==============================] - 46s 2s/step - loss: 0.0190 - val_loss: 0.0193
<tensorflow.python.keras.callbacks.History at 0x7f5ee1851090>
```

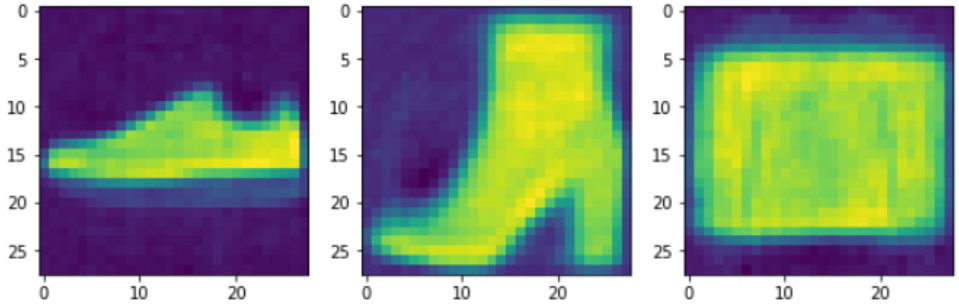
میزان loss و val_loss نیز به کمترین مقدار خود یعنی "۰.۰۱۹۰" و "۰.۰۱۹۳" رسیده‌اند.

پیش‌بینی بر روی داده‌ی ولیدیشن

ابتدا تصاویر واقعی برخی داده‌ها را نمایش می‌دهیم:



حال داده‌های بالا را با استفاده از اتوانکدر طراحی شده، پیش‌بینی می‌کنیم:



همانطور که مشخص است، اتوانکدر ساخته شده با epoch۲۰ ، قادر به بازسازی تصاویر ورودی به خوبی خودشان می‌باشد.

بخش دوم: Modify Image Reonstruction

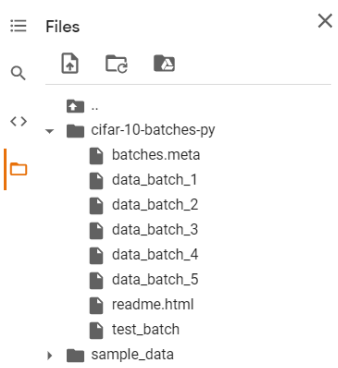
در این بخش می‌خواهیم یک اتوانکدر stack شده را برای بازسازی تصاویر طراحی کنیم. برای اینکار طربق گفته‌ی صورت سوال، از دیتاست CIFAR_10 استفاده می‌کنیم. این دیتاست به صورت رایگان در این لینک قرار گرفته است. دیتاست CIFAR_10، شامل ۶۰۰۰۰ تصویر رنگی در ابعاد ۳۲*۳۲ است. داده‌های آن به دو بخش آموزش(۵۰۰۰۰) و تست(۱۰۰۰۰) تقسیم شدند. این دیتاست دارای ۱۰ کلاس زیر می‌باشد: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck

به دلیل اینکه حجم داده‌های cifar ۱۷۰ مگابایت است و آپلود آن‌ها در colab زمان زیادی را می‌برد، می‌توان با استفاده از تکه کد زیر فایل این دیتاست را مستقیما از سایت toronto دانلود و در سایت colab، آپلود کرد.

```python
from urllib.request import urlretrieve
from os.path import isfile, isdir
from tqdm import tqdm
import tarfile

cifar10_dataset_folder_path = 'cifar-10-batches-py'

class DownloadProgress(tqdm):
    last_block = 0

    def hook(self, block_num=1, block_size=1, total_size=None):
        self.total = total_size
        self.update((block_num - self.last_block) * block_size)
        self.last_block = block_num

"""
    check if the data (zip) file is already downloaded
    if not, download it from "https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz" and save as cifar-10-python.tar.gz
"""
if not isfile('cifar-10-python.tar.gz'):
    with DownloadProgress(unit='B', unit_scale=True, miniters=1, desc='CIFAR-10 Dataset') as pbar:
        urlretrieve(
            'https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz',
            'cifar-10-python.tar.gz',
            pbar.hook)

if not isdir(cifar10_dataset_folder_path):
    with tarfile.open('cifar-10-python.tar.gz') as tar:
        tar.extractall()
        tar.close()
```

```
CIFAR-10 Dataset: 171MB [00:02, 61.4MB/s]
```

در این صورت، فایل cifar-10-batches-py در قسمت فایل‌ها پدیدار می‌شود.



پیش‌پردازش

حال می‌توان این فایل را از طریق مسیری که در خود سایت cifar توضیح داده شده است، باز کنیم.

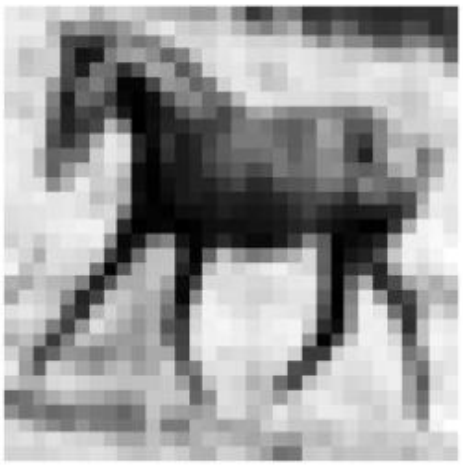
And a python3 version:

```
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

برای سادگی کار، بعد چهارم که نمایش دهنده‌ی رنگ تصاویر است را حذف کرده و به جای آن، فرمت سیاه و سفید را پیاده سازی می‌کنیم. پس از پیش‌پردازش و سیاه سفید کردن داده‌ها، فرمت x و y به شرح زیر می‌گردد:

```
X:  (50000, 1024)
Y:  (50000,)
```

با توجه به صورت سوال، مجبوریم برای سادگی کار تنها با کلاس هفتم یعنی کلاس اسب‌ها، عملیات را انجام دهیم. برای اینکار از np.where استفاده می‌کنیم و جایی که y برابر با ۷ باشد را جزو حوزه‌ی x انتخاب می‌کنیم. قبل از شروع کار، تصویری از قسمت اسب‌های این دیتاست را در ادامه مشاهده می‌کنید: تصویر ۵۳ام



Dataset Estimator

قبل از اینکه بخواهیم مدل را با استفاده از Tensorflow، بسازیم، لازم است با استفاده از dataset estimator، شبکه را تغذیه کنیم. برای اینکار به تکه کد زیر احتیاج داریم.

```
dataset = tf.data.Dataset.from_tensor_slices(x).repeat().batch(batch_size)
```

باید دقت شود که در اینجا، x یک محل نگهدارنده با فرمت [None, n_input] است.

Tf.data.dataset، APIای است که یک pipeline موثر و توصیفی را پدید می‌آورد. برای پدید آوردن آن، مراحل زیر را طی می‌کند:

۱. از داده‌های ورودی، یک دیتاست منبع درست می‌کند.

۲. تبدیلاتی بر روی دیتاست انجام می‌دهد تا بتواند داده‌ها را پیش‌پردازش کند.

۳. همین عملیات را بر روی دیتاست تکرار می‌کند و عناصر را پردازش می‌کند.

بنابراین برای تکرار عملیات ذکر شده، نیاز به یک iterator داریم. با استفاده از آن، داده‌ها به داخل pipeline ورود پیدا می‌کنند.

```
iter = dataset.make_initializable_iterator() # create the iteratorfeatures = iter.get_next()
```

حال که دو مورد ذکر شده را فراهم کردیم، کافی است به batch_size مقدار دهیم و بررسی کنیم آیا تصویر ۵۳ام که در گذشته نمایش دادیم با تصویر تولیدی پس از انجام مراحل ذکر شده دارای ظاهری مشابه است؟

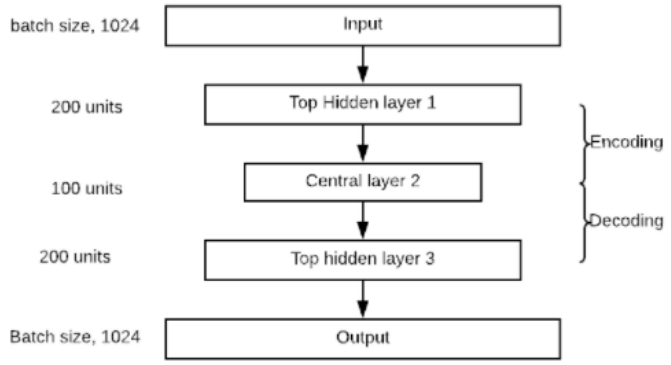
```
(1, 1024)
```

- Batch_size را با عدد ۱ مقداردهی می‌کنیم، زیرا تنها می‌خواهیم یک تصویر از دیتاست را به شبکه دهیم.

(1,24) به این معناست که تنها یک تصویر با ۱۰۲۴ تصویر دیگر، تغذیه می‌شود.

ساخت شبکه

با توجه به صورت سوال، باید شبکه‌ای با معماری زیر طراحی کنیم:



Building the network for Autoencoder

برای ساخت یک اتوانکدر لازم است قدم‌های زیر را برداریم:

۱. تعریف پارامترها
۲. تعریف لایه‌ها
۳. تعریف ساختار
۴. تعریف بهینه‌ساز
۵. اجرای مدل
۶. ارزیابی مدل

در بخش گذشته، pipelineای طراحی کردیم که مدل را تغذیه می‌کرد. در این بخش می‌خواهیم اتوانکدری طراحی کنیم که دارای ۴ لایه است (مطابق تصویر). برای اینکار از تکنیکی به نام Xavier initialization استفاده می‌کنیم. در این تکنیک، وزن‌های اولیه را برابر با واریانس ورودی و خروجی قرار می‌دهیم. در نهایت از تابع فعالیت ElU برای آموزش و تابع هزینه‌ی L2 برای ارزیابی استفاده می‌کنیم.

۱) تعریف پارامترها

پارامترها در واقع تعداد نورون‌ها در هر لایه، نرخ یادگیری و hyperparameter یک رگولاریزیشن را نمایش می‌دهند. لازم به ذکر است، نرخ یادگیری را ۰.۰۱ و رگولاریزیشن L2 را ۰.۰۰۰۱ قرار دادیم.

۲) تعریف لایه‌ها

تمامی پارامترهای لایه‌ی dense به صورت بسته بندی شده در dense_layer قرار گرفتند. تنها کافی است تابع فعالیت، تکنیک Xavier و رگولاریزیشن را تعریف کنیم.

۳) تعریف ساختار

لایه‌ی اول، وظیفه‌ی محاسبه‌ی ضرب داخلی ماتریس ویژگی‌های ورودی و ماتریسی که شامل ۲۰۰ وزن است را دارد. پس از محاسبه‌ی ضرب داخلی، خروجی وارد تابع فعالیت ELU می‌گردد و خروجی آن، ورودی لایه‌ی بعدی به حساب می‌آید. ضرب ماتریسی دارای فرمت یکسانی برای هر لایه دارد، زیرا از یک تابع فعالیت یکسان برای تمام لایه‌های استفاده می‌شود. تنها باید توجه شود که بر روی خروجی لایه‌ی آخر، هیچ تابع فعالیتی اعمال نمی‌گردد، زیرا این خروجی، نتیجه‌ی تصویر بازسازی شده است.

۴) تعریف بهینه ساز

در این مرحله، از MSE برای تابع هزینه استفاده می‌کنیم. این تابع، تفات میان مقدار پیش‌بینی شده و مقدار واقعی را به عنوان خطا باز می‌گرداند. در اینجا، labelها، ویژگی‌های ما هستند زیرا مدل سعی در بازسازی ورودی دارد. حال برای بهینه کردن تابع هزینه، نیاز به یک optimizer داریم. می‌توان از Adam برای محاسبه‌ی گرادیان استفاده کرد. هدف آن، مینیمم سازی تابع هزینه یا خطا است.

نکته: تعیین تعداد batchها، یعنی تعداد تصاویری که می‌خواهیم به pipeline به عنوان ورودی دهیم، قابل محاسبه است. می‌توان طول دیتاست را بر سایز batchها تقسیم کرد و تعداد batchها را بدست آورد. در اینجا به دلیل اینکه طول batchها ۱۰۰ بود، تعداد آن‌ها ۵۰ محاسبه شد.

۵) اجرای مدل

در اینجا، مدل را با epoch ۱۰۰، آموزش می‌دهیم. تنها تفاوتی که این آموزش با آموزش‌های قبل دارد، در pipe کردن داده قبل از شروع آموزش است. در این صورت، مدل سریعتر آموزش می‌بیند.

```
Training...
(150, 1024)
0 Train MSE: 2804.4246
10 Train MSE: 1579.918
20 Train MSE: 1367.6409
30 Train MSE: 1196.495
40 Train MSE: 1134.7667
50 Train MSE: 1229.7611
60 Train MSE: 1338.7965
70 Train MSE: 1199.9559
80 Train MSE: 1252.8225
90 Train MSE: 1194.6837
Model saved in path: ./model.ckpt
```

۶) ارزیابی مدل

برای انجام این مرحله، مجموعه داده‌های تست را از cifar، فراخوانی می‌کنیم و برای مثال، تصویر ۱۳ ام را از آن نمایش می‌دهیم:

برای ارزیابی مدل، از پیکسل‌های این تصویر استفاده می‌کنیم. درواقع می‌خواهیم ببینیم آیا انکودر می‌تواند تصویر مشابه را پس از کوچک کردن ۱۰۲۴ پیکسل، بازسازی کند. برای اینکار تابعی با نام reconstruct_image معرفی می‌کنیم که دو آرگومان df و image_number را دریافت می‌کند. (توجه شود، این تابع برای اسب‌ها طراحی شده است. بنابراین نتیجه‌ی بهتری بر روی کلاس اسب‌ها دارد و برای بازسازی تصاویر کلاس‌های دیگر بهتر است توابع دیگری معرفی شود.)

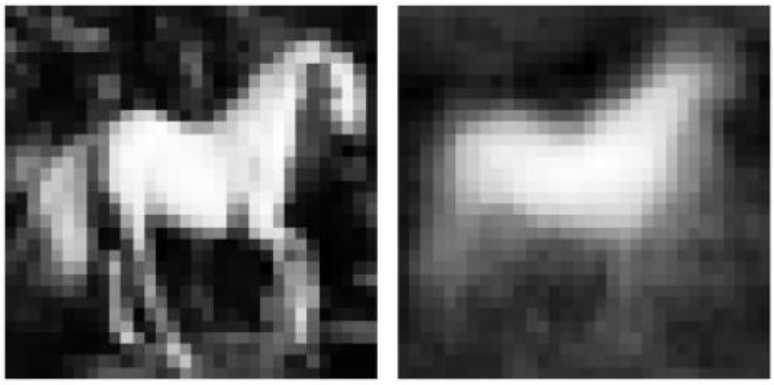
Df: برای فراخوانی داده‌های تست معرفی شده است.

Image_number: تعداد و شماره‌ی تصویری را نمایش می‌دهد که قرار است نمایش داده شود.

این تابع کار خود را در سه بخش انجام می‌دهد:

- تصویر را به ابعاد مناسب ۱ و ۱۰۲۴، reshape می‌کند.
- تصویری که مدل تا به حال ندیده را در اختیار مدل قرار می‌دهد و آن تصویر را انکود و دیکد می‌کند.
- در نهایت نیز تصویر واقعی و یازسازی شده را چاپ می‌کند.

```
INFO:tensorflow:Restoring parameters from ./model.ckpt
Model restored.
(1, 1024)
```



از نتیجه مشخص است که توانسته shape تصویر را باطسازی کند. بنابراین اتوانکدر ذکر شده به درستی ارزیابی را انجام می‌دهد.

بخش سوم: Image Denoising

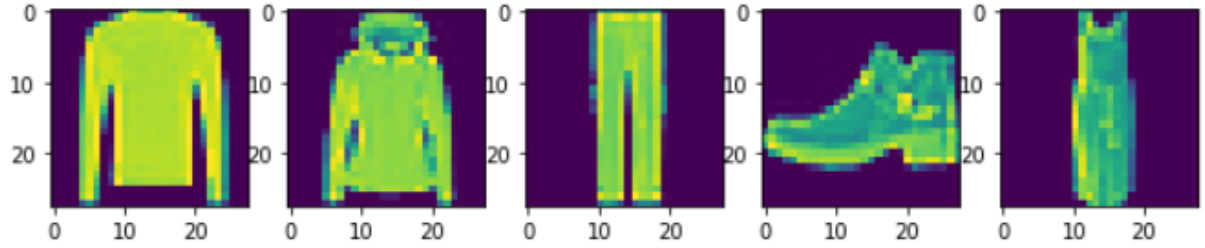
یکی دیگر از کاربردهای اتوانکدرها، حذف نویز از تصویر ورودی شبکه است. برای اینکار از شبکه‌های کانولوشنی نیز کمک می‌گیریم. ولی قبل از طراحی مدل باید نویز به تصاویر ورودی وارد کنیم.

تصاویر نویزی

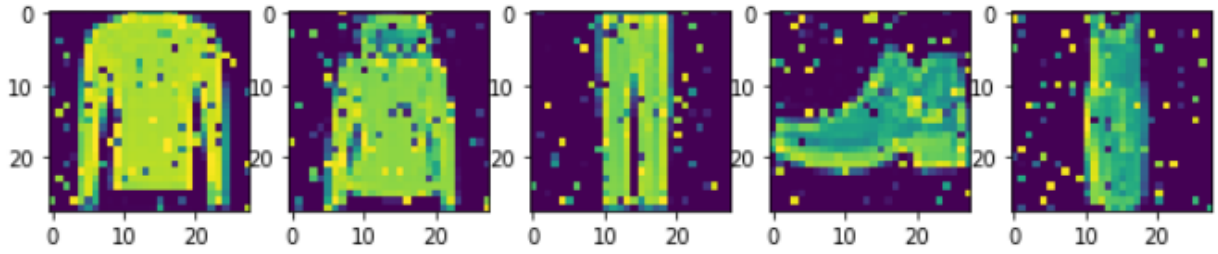
نویز انواع متفاوتی دارد که در ادامه به ۴ مورد از آن‌ها اشاره شده است:

- نویز نمک فلفلی
- نویز گوسی
- نویز دوره‌ای / تناوبی[1]
- نویز خال خال[2]

برای حل این بخش از نویز نمک فلفلی استفاده می‌کنیم. این نویز توزیع ناگهانی و سریعی را در سیگنال تصویر اعمال می‌کند. در ادامه تصاویری از دیتاست را مشاهده می‌کنید که به صورت معمولی و بدون اضافه کردن نویزی به آن‌ها نمایش داده می‌شوند.



حال به تصاویر بالا نویز نمک فلفلی اضافه می‌کنیم. در این صورت تصاویر نویز دار به شرح زیر می‌شوند.



[1] Periodic
[2] Speckle

طراحی معماری اتوانکدر

- Encoding

همانطور که در صورت تمرین مطرح شده است، این لایه باید دارای ۳ لایه‌ی کانولوشن و ۳ لایه‌ی پولینگ باشد که روی یکدیگر stacked شدند. برای طراحی این لایه از Relu به عنوان تابع فعالیت استفاده می‌شود و padding به صورت "مشابه" نگهداری می‌شود. نقش max pooling نیز کاهش بعد تصاویر ورودی است.

- Decoding

همانند لایه‌ی encoding، لایه‌های کانولوشن به ترتیب برعکس قرار می‌گیرند (با همان بعد). بجای استفاده از سه لایه‌ی pooling Max از سه لایه‌ی Upsampling در مقابل downsampling در انکدر استفاده می‌شود. نقش این لایه Upsample کردن بردار ورودی به رزولوشن بالاتر است.

معماری معرفی شده به صورت زیر پیاده‌سازی می‌شود:

```
#input layer
input_layer = Input(shape=(28, 28, 1))

#encoding layer
encoded_layer1 = Conv2D(64, (3, 3), activation='relu', padding='same')(input_layer)
encoded_layer1 = MaxPool2D( (2, 2), padding='same')(encoded_layer1)
encoded_layer2 = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded_layer1)
encoded_layer2 = MaxPool2D( (2, 2), padding='same')(encoded_layer2)
encoded_layer3 = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded_layer2)
latent_view    = MaxPool2D( (2, 2), padding='same')(encoded_layer3)

#decoding layer
decoded_layer1 = Conv2D(16, (3, 3), activation='relu', padding='same')(latent_view)
decoded_layer1 = UpSampling2D((2, 2))(decoded_layer1)
decoded_layer2 = Conv2D(32, (3, 3), activation='relu', padding='same')(decoded_layer1)
decoded_layer2 = UpSampling2D((2, 2))(decoded_layer2)
decoded_layer3 = Conv2D(64, (3, 3), activation='relu')(decoded_layer2)
decoded_layer3 = UpSampling2D((2, 2))(decoded_layer3)
output_layer   = Conv2D(1, (3, 3), padding='same')(decoded_layer3)

#compile
model_2 = Model(input_layer, output_layer)
model_2.compile(optimizer='adam', loss='mse')

#summary
model_2.summary()
```

Summary این مدل نیز به شرح زیر گزارش می‌گردد:

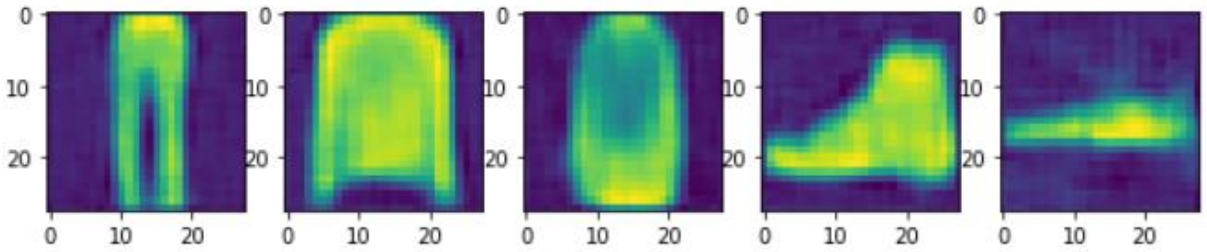
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 16)          4624
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 16)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 16)          2320
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 8, 8, 16)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 32)          4640
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 1)         577
=================================================================
Total params: 49,761
Trainable params: 49,761
Non-trainable params: 0
_________________________________________________________________
```

حال مدل ساخته شده را به وسیله‌ی early stopping، آموزش می‌دهیم. تعداد epochها را به بالاترین مقدار ممکن برای دستیابی به نتیجه‌ی بهتر، افزایش می‌دهیم.

```
Epoch 1/10
24/24 [==============================] - 202s 8s/step - loss: 0.0291 - val_loss: 0.0273
Epoch 2/10
24/24 [==============================] - 202s 8s/step - loss: 0.0256 - val_loss: 0.0245
Epoch 3/10
24/24 [==============================] - 202s 8s/step - loss: 0.0235 - val_loss: 0.0229
Epoch 4/10
24/24 [==============================] - 203s 8s/step - loss: 0.0221 - val_loss: 0.0218
Epoch 5/10
24/24 [==============================] - 202s 8s/step - loss: 0.0212 - val_loss: 0.0211
Epoch 6/10
24/24 [==============================] - 202s 8s/step - loss: 0.0205 - val_loss: 0.0206
Epoch 7/10
24/24 [==============================] - 207s 9s/step - loss: 0.0199 - val_loss: 0.0199
Epoch 8/10
24/24 [==============================] - 197s 8s/step - loss: 0.0196 - val_loss: 0.0194
Epoch 9/10
24/24 [==============================] - 198s 8s/step - loss: 0.0190 - val_loss: 0.0190
Epoch 10/10
24/24 [==============================] - 199s 8s/step - loss: 0.0186 - val_loss: 0.0189
```

پیش‌بینی مدل

در ادامه تصویری از پیش‌بینی را مشاهده می‌کنید که از مدل طراحی شده برآمده است. درست است که حوالی تصویر را درست پیش‌بینی کرده است ولی برای دریافت دقت بالاتر و نتیجه‌ای بهتر باید epoch آموزش را بیشتر کنیم. یعنی به جای ۱۰ مورد، ۵۰۰ تا ۱۰۰۰ مورد را برای پیش‌بینی مد نظر بگیریم. به دلیل نبود GPU مناسب و کمبود ظرفیت Colab GPU، ما این تکرار را به ۱۰ مورد کاهش دادیم.



بخش چهارم: Seq to Seq Prediction

مقدمه

در بخش‌های گذشته، ورودی را تصویری دو بعدی تعریف می‌کردیم. در این بخش به دلیل کاربرد اتوانکدرها در داده‌های ترتیبی، دیگر تصاویر دو بعدی را به عنوان ورودی نمی‌دهیم. بلکه سری‌های زمانی یا متنی را با یک بعد به ورودی خواهیم داد. این کاربرد را می‌توان در ترجمه‌ی ماشینی به کار برد و به جای استفاده از CNNها، می‌توان از LSTMها که مناسب برای داده‌های ترتیبی هستند استفاده کرد.

معماری اتوانکدرها

این معماری دارای یک encoder برای encode کردن توالی منبع و یک decoder برای decode کردن توالی منبع encode شده به توالی هدف. قبل از شرح گسترده‌ی این دو لایه، لازم است کارکرد LSTM را کمی توضیح دهیم. LSTM، یک شبکه‌ی RNN است که شامل یک سری گیت‌های داخلی می‌باشد. ولی برخلاف دیگر RNNها، گیت‌های داخلی شبکه، به مدل این اجازه را می‌دهند که با موفقیت آموزش ببیند و این آموزش را از طریق BPTT انجام می‌دهد. همین عمل باعث می‌شود از مشکل ناپدید شدن گرادیان دوری کند. می‌توان واحدهایی در LSTM، تحت عنوان حافظه، معرفی کرد. که هرکدام دارای یک حافظه‌ی داخلی هستند. لازم به ذکر است، با استفاده از keras، می‌توان هم به state خارجی و هم به state فعلی LSTM، دسترسی پیدا کرد.

حال به بررسی معماری اتوانکدر می‌پردازیم:

- Encoder

این لایه، یک توالی را به عنوان ورودی دریافت می‌کند و state فعلی LSTM را به عنوان خروجی باز می‌گرداند.

- Decoder

این لایه، یک توالی و stateای که در LSTM، encode شده را به عنوان ورودی می‌گیرد. سپس توالی decode شده را به عنوان خروجی باز می‌گرداند.

← لازم به ذکر است: در طول پیاده‌سازی، stateهای حافظه و پنهان را نگهداری می‌کنیم تا بتوانیم در زمان پیش‌بینی توالی‌های unseen، از آن‌ها استفاده کنیم.

پیاده‌سازی

برای شروع کار، نیزا به یک دیتاست برای پیاده‌سازی معماری‌های ذکر شده بر روی آن داریم. می‌خواهیم یک دیتاست توالی را که شامل توالی‌های رندوم با طول ثابت هستند را با استفاده از یک تابع، تولید کنیم.

X1، نشان دهنده‌ی توالی ورودی است که شامل شماره‌های رندوم است.

X2، نشان دهنده‌ی توالی padded است که از آن به عنوان seed برای تولید دیگر عناصر توالی استفاده می‌شود.

y، نمایش دهنده‌ی توالی هدف یا توالی واقعی است.

عملیات ذکر شده با استفاده از تابع dataset_preparation انجام می‌شود. در ادامه، شکل ظاهری داده‌های تولید شده را مشاهده می‌کنید:

```
Shapes:  (100000, 6, 51) (100000, 3, 51) (100000, 3, 51)
first categorically encoded input sequence looks like:
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
      dtype=float32)
```

حال می‌خواهیم معماری مدل را با استفاده از keras پیاده‌سازی کنیم. این عمل را با استفاده از تابع define_models انجام می‌دهیم:

```python
def define_models(n_input, n_output):
    #encoder architecture -> I=sequence - O=encoder states
    encoder_inputs = Input(shape=(None, n_input))
    encoder = LSTM(128, return_state=True)
    encoder_outputs, state_h, state_c = encoder(encoder_inputs)
    encoder_states = [state_h, state_c]

    #encoder-decoder architecture -> I=seed sequence - O=decoder states, decoded output
    decoder_inputs = Input(shape=(None, n_output))
    decoder_lstm = LSTM(128, return_sequences=True, return_state=True)
    decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_states)
    decoder_dense = Dense(n_output, activation='softmax')
    decoder_outputs = decoder_dense(decoder_outputs)
    model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

    #decoder model -> I=current states + encoded sequence - O=decoded sequence
    encoder_model = Model(encoder_inputs, encoder_states)
    decoder_state_input_h = Input(shape=(128,))
    decoder_state_input_c = Input(shape=(128,))
    decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
    decoder_outputs, state_h, state_c = decoder_lstm(decoder_inputs, initial_state=decoder_states_inputs)
    decoder_states = [state_h, state_c]
    decoder_outputs = decoder_dense(decoder_outputs)
    decoder_model = Model([decoder_inputs] + decoder_states_inputs, [decoder_outputs] + decoder_states)

    return model, encoder_model, decoder_model
```

در ادامه summary بخش‌های مختلف مدل طرح شده را مشاهده می‌کنید.

- Encoder:

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, 51)]        0
_________________________________________________________________
lstm (LSTM)                  [(None, 128), (None, 128) 92160
=================================================================
Total params: 92,160
Trainable params: 92,160
Non-trainable params: 0
_________________________________________________________________
```

- Decoder:

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, None, 51)]   0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 128)]        0
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 128)]        0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, None, 128),  92160       input_2[0][0]
                                                                 input_3[0][0]
                                                                 input_4[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, None, 51)     6579        lstm_1[1][0]
==================================================================================================
Total params: 98,739
Trainable params: 98,739
Non-trainable params: 0
__________________________________________________________________________________________________
```

- Autoencoder:

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, None, 51)]   0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None, 51)]   0
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 128), (None, 92160       input_1[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, None, 128),  92160       input_2[0][0]
                                                                 lstm[0][1]
                                                                 lstm[0][2]
__________________________________________________________________________________________________
dense (Dense)                   (None, None, 51)     6579        lstm_1[0][0]
==================================================================================================
Total params: 190,899
Trainable params: 190,899
Non-trainable params: 0
__________________________________________________________________________________________________
```

این summary، شامل اطلاعاتی همچون، تعداد پارامترها، پارامترهای آموزش دیده و پارامترهای آموزش ندید است. که همانطور که مشخص است در هیچ کدام، پارامترهایی نداریم که آموزش ندیده باشند.

حال نوبت به آن رسیده است که داده‌های موجود را با مدل طراحی شده، آموزش دهیم.

```
autoencoder.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
autoencoder.fit([inputs, seeds], outputs, epochs=1)
```

```
3125/3125 [==============================] - 29s 8ms/step - loss: 1.4507 - acc: 0.5511
<tensorflow.python.keras.callbacks.History at 0x7fe1fc548f50>
```

میزان دقت آموزش، ۵۵٪ است. این دقت را می‌توان با افزایش تعداد epochها، افزایش داد.

حال که آموزش را نیز انجام دادیم، می‌توانیم پیش‌بینی را انجام دهیم:

برای انجام اینکار، ابتدا بردارها را onehot می‌کنیم و سپس با تعریف تابعی به نام predict_sequence، پیش‌بینی را انجام می‌دهیم.

```
Input Sequence=[14, 19, 20, 38, 27, 39] SeedSequence=[20, 19, 14], PredictedSequence=[20, 19, 14]

Input Sequence=[49, 44, 17, 5, 5, 20] SeedSequence=[17, 44, 49], PredictedSequence=[17, 44, 49]

Input Sequence=[1, 40, 7, 47, 27, 33] SeedSequence=[7, 40, 1], PredictedSequence=[7, 40, 1]

Input Sequence=[4, 43, 37, 19, 15, 45] SeedSequence=[37, 43, 4], PredictedSequence=[37, 43, 4]

Input Sequence=[35, 40, 18, 18, 37, 50] SeedSequence=[18, 40, 35], PredictedSequence=[18, 40, 35]
```

همانطور که مشخص است، پیش‌بینی ۱۰۰٪ درست انجام شده است. بنابراین اتوانکدر طراحی شده، درست کار می‌کند.

بخش پنجم: منابع و مراجع

https://www.kaggle.com/shivamb/how-autoencoders-work-intro-and-usecases#2.3-UseCase-3:-Sequence-to-Sequence-Prediction-using-AutoEncoders

https://machinelearningmastery.com/develop-encoder-decoder-model-sequence-sequence-prediction-keras/

https://github.com/deep-diver/CIFAR10-img-classification-tensorflow/blob/master/CIFAR10_image_classification.ipynb

https://www.guru99.com/autoencoder-deep-learning.html

توجه شود: در این تمرین از سایت‌های kaggle و guru به فراوانی استفاده شد و در برخی موارد، کدهایی را از آن‌ها به عاریه گرفتم. منتها در تمامی موارد ذکر شده، سعی کردم خط به خط کد را متوجه شوم و گزارش آن را به صورت کامل پیاده کنم.