Assignment NO.1 Solutions

Deep learning | winter 1400 | Dr.Mohammadi

Teacher Assistant:

Shabnam Ezatzadeh

Student name : **Amin Fathi**

Student id : **400722102**

Problem 1

Learning algorithm

مسایل یادگیری ماشین ، مانند مسایل برنامه نویسی نیستند که از ۰ تا ۱۰۰ و به صورت scratch بتوانیم آن ها را خودمان پیاده سازی و برنامه نویسی کنیم و و ما اغلب نمیتوانیم به ماشین(کامپیوتر) نشان بدهیم که چگونه از ورودی ها به خروجی برسد (و بر اساس چه رابطه ای) . در این مسایل ما قصد و توان پیاده سازی یک برنامه "صریح" را نداریم(مانند برنامه های کد نویسی) ، در عوض می توانیم یک برنامه "انعطلف پذیر" را تعریف می کنیم که رفتارش با تعدادی پارامتر تعیین می شود و سپس با استفاده مجموعه ای داده ، پارامتر های بهینه را برای این برنامه انعطاف پذیر به دست می آوریم به طوری که برنامه ما بهترین عملکرد مد نظر ما برای مساله را داشته باشد . در واقع مدل ، برنامه ای است که پارامتر های آن پیدا شده و ثابت باشند .

به مجموعه ای از مدل ها که تنها تفاوتشان در مقدار پارامتر هایشان است ، خانواده مدل ها گویند . و به ابربرنامه یا همان روش و اسلوب و الگوریتمی که با استفاده از دادگان و دیتا پارامتر های برنامه را تعیین میکند را الگوریتم یادگیری یا learning algorithm گویند مانند : SVM , Linear Regression و ... .

برای انتخاب الگوریتم یادگیری مناسب ابتدا باید مساله و ماهیت ورودی ها و خروجی ها و خانواده مدل را به صورت دقیق مشخص کنیم سپس الگوریتم یادگیری مناسب اعمال شود.

Objective function

از آنجا که یادگیری ماشین را میتوان یادگیری تو تجربه و نوعی از مسایل بهینه سازی تعریف کرد ، با تعریف کردن مسایل یادگیری ماشین به صورت مسایل یادگیری بهینه سازی ، لازم است ، به طور ریاضی ، راه حل "بهینه " و مناسب را از میان راه حل های مختلف را برای آن به دست آورد . وظیفه سنجش کیفیت راه حل ها به جهت انتخاب بهترین راه حل برای مسایل یادگیری ماشین (بهینه سازی پاسخ مساله یا همان بهینه سازی مدل) بر عهده Objective function می باشد . ورودی این تابع پارامتر های مساله (اینجا پارامتر های مدل) می باشد

به طور معمول و قراردادی هر چه مقدار خروجی Objective function کمتر باشد ، مدل ما بهتر است و پاسخ بهینه تر ، البته میتوان Objective function را به شیوه ای در نظر گرفت که هر چه مقدارش بیشتر باشد ، جواب بهینه تر باشد.

در مسایل یادگیری ماشین توابع loss function یک object function می باشد که هر چه خروجی اش کمتر باشد مدل ما بهتر است و هدفش یافتن بهینه ترین پارامتر های مدل است ، به عنوان مثال تابع ضرر مربع خطا که هر چه مقدارش کمتر، مدل بهتر است .

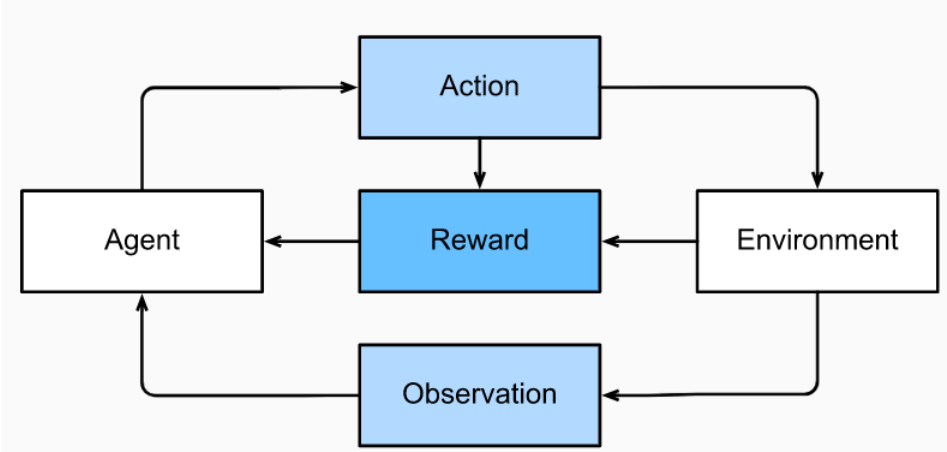
4D tensors vs. 4-dimensional vector

بردار 4 بعدی گونه ای از آبجکت های ریاضی ایستا است و در واقع مانند یک ظرف نگهداری برای داده هایی با اندازه ثابت 4 بعدی است ، به طور واضح تر می توان گفت اندازه داده هایی که در بردار 4 بعدی ذخیره میکنیم ثابت است ، مثلا بردار 4 بعدی به ابعاد a*b*c*d حامل داده ای به اندازه حاصل ضرب بزرگی ابعاد (a*b*c*d) است و این اندازه ثابت است . ثابت بودن اندازه بردار ها در جهان یادگیری ماشین و عمیق چندان نکته خوبی به حساب نمی آید ، چرا که ما با داده هایی با طول های متفاوت مواجهیم (فرض کنید قصد دارید تصاویر سرطانی به عنوان داده آموزشی به شبکه عصبی تحویل دهید ، مسلما یک دست کردن ابعاد همه تصاویر با تکنیک های crop و resize ممکن است بخش خوبی از ویژگی های مهم تصاویر را از بین ببرد) برای حل این مشکل بدون تلاش سهوی برای از بین بردن ویژگی داده ها از تنسور ها استفاده میکنیم . تنسور ها هم همانند بردار ها گونه ای از آبجکت های ریاضی هستند ، اما آبجکت های ریاضی پویا و داینامیک و برای مدیریت کردن داده ها با طول ها مختلف به کار میروند . تنسور ها به نوعی حالت عمومی برای آرایه های n بعدی به شمار میروند ، تنسور رنک0 را میتوان اسکالر نامید ، تنسور رنک 1 بردار است و تنسور رنک 2 همان ماتریس یا بردارد 2 بعدی است .اما تفاوت اصلی تنسور با بردار در دینامیک بودن و پیروی کردن تنسور از قوانین انتقالی بین ابعاد پایه (transformation laws)است (برای مشاهده مثال هایی از این مزیت تنسور ها به این لینک و این لینک مراجعه شود)

اما به جهت پاسخگویی دقیقتر به سوال با توجه به بخش 2.3.2.1 کتاب مرجع باید گفت که vector یک تنسور با رنک 1 است و بعد در vector اصلاحا نشان دهنده اندازه آن است (اندازه در معنای مقدار داده ای که ذخیره می کند) ولی بعد در تنسور (یا همان رنک) نشان دهنده چند محوره بودن داده می باشد ؛ طبق این تعریف vector 4d یک تنسور یک محوره است با دیتایی به اندازه و بزرگی 4 در حالی که tensor 4d در واقع یک آبجکت ریاضی و جبری 4 محوره می باشد.

Reinforcement Learning

یادگیری تقویتی برای مسائلی به کار می رود که در آن ها عامل با محیط هم در تعامل است و از آن اثر می پذیرد مانند رباتیک و هوش مصنوعی در بازی های ویدیویی . در یادگیری تقویتی ، عامل در مرحله (از لحاظ زمانی) مشاهداتی(observation) از محیط(Enviroment) را دریافت می کند و با انجام عملی (action) بر محیط تاثیر گذاشته و همچنین پاداش (reward) متناسب با آن عمل در آن محیط را دریافت می کند و سپس عامل مشاهده بعدی را دریافت می کند و عمل بعدی را انتخاب کرده و پاداش بعدی را هم دریافت می کند و ... . این فرایند در شکل زیر نمایش داده شده است .



سیاست یا خط مشی (policy) در یادگیری توقویتی در واقع بیانگر پروسه تصمیم گیری برای انتخاب عمل (action) با توجه به مشاهدات محیطی و پاداش ها برای عامل است . میتوان هدف یادگیری تقویتی را ایجاد یک policy خوب برای مساله در نظر گرفت.

در جدول زیر تفاوت های یادگیری تقویتی (Reinforcement learning) – یادگیری بی ناظر (Unsupervised learning) – یادگیری با ناظر (Supervised learning) را مشاهده می کنیم .

	Supervised	Unsupervised	Reinforcement
تعریف	یادگیری با استفاده از داده های برچسپ خورده و با راهنمایی و نظارت آموزشی	یادگیری با استفاده از دیتا های بدون برچسپ و بدون راهنمایی برای آموزش	روی تعامل با محیط کار میکند
نوع داده	داده برچسپ دار	داده بدون برچسپ	بدون داده های ازپیش تعیین شده
نوع مسایلی که با ان ها سر وکار دارد	رگرسیون و طبقه بندی	Association (وابستگی ها)و خوشه بندی (کلاسترینگ)	Exploitation or Exploration
الگوریتم ها	Linear Regression, Logistic Regression, SVM, KNN etc.	K – Means, C – Means	Q – Learning, SARSA
هدف نهایی	محاسبه خروجی برای مساله	یافتن الگو های مخفی بین داده های برچسپ نخورده	یافتن دنباله ای action های مناسب یا همان policy

منبع :

۱- کتاب درسی

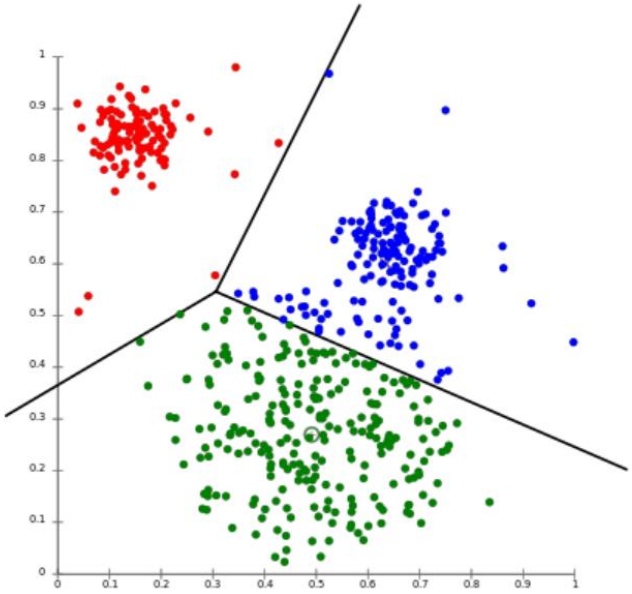
۲ - Supervised vs Unsupervised vs Reinforcement - AITUDE

۳ - Objective Functions in Machine Learning (kronosapiens.github.io)

Problem2

همانطور که در قسمت قبل توضیح داده شد ، unsupervised learning یا یادگیری بی ناظر برای پیدا کردن الگو های مخفی بین داده هایی که برچسپ ندارند به کار می رود و سوالی که ما در این قسمت با آن رو به رو هستیم یک سوال clustering یا خوشه بندی است که با استفاده از روش یادگیری بی ناظر قرار است به آن پاسخ دهیم .

فرض کنیم در این سوال ما دادگان ما خبر هایی هستند که به طور مثال در یک بازه زمانی مشخص (مثلا بین سال های ۱۳۸۰ تا ۱۴۰۰) از سایت های خبری مشخصی جمع آوری شده اند ، این اخبار هر کدام نویسنده ، زمان انتشار ، سایت منتشر کننده ، تیتر عنوان خبر مخصوص به خود را دارند ، داده ها را پس از پیش پردازش و مرتب کردن برای خوشه بندی تحویل یک الگوریتم خوشه بندی می دهیم (به طور مثال k-means) . الگوریتم های خوشه بندی با توجه به شباهت میان ویژگی های داده ها آن ها را به چند خوشه تقسیم می کنند که در هر خوشه دیتا هایی به مراتب شبیه تر به هم حضور دارند ؛ در شکل زیر خوشه بندی یک دیتای فرضی به سه خوشه را مشاهده می کنید



شمایی از الگوریتم k-means

فرض کنید پس از اجرای الگوریتم خوشه بندی بر روی دادگان (فرض کنید هدف نهایی ما خوشه بندی به ۱۰ خوشه می باشد) ؛ ۱۰ کلمه پر تکرار هر خوشه به شرح زیر است :

خوشه اول : فوتبال ، سرمربی ، کنفرانس خبری ، باخت ، برد ، حذف ، قهرمان ، داور ، نتیجه ، هواداران

خوشه دوم : دادگاه ، قاضی ، عدالت ، معترضان ، زندان ، محاکمه ،قانون ، قوه قضاییه ، اعدام ، جریمه

خوشه سوم : مالیات ، بازار آزاد ، ارزش ریال ، دلار ، تورم ، نقدینگی ، تحریم ، بانک مرکزی ، بیمه ، بیکاری

خوشه چهارم : برجام ، آمریکا ، شورای امنیت ، ژنو ، رییس جمهور ، مذاکرات ، تحریم ، ۵+۱ ، بین الملل ، همکاری

خوشه پنجم : انتخابات ، ملی ، ایران ، شورا ، مجلس ، احزاب ، دموکراسی ، جامعه مدنی ، پاستور ، تظاهرات

خوشه ششم : بیکاری ، طلاق ، زندان ، معتاد ، فقر ، آسیب ، اجتماعی ، زنان ، اقشار ، کارگر

خوشه هفتم : موشک ، دفاع ، جنگ ، خاورمیانه ، نظامیان ، مانور ، رهبری ، فرماندهان ، ایران ، سامانه

خوشه هشتم : صنعت ، کارخانه ، gdp ، توسعه ، صادرات ، مالیات ، کارگر ، رشد ، اتاق بازرگانی ، تولید

خوشه نهم : نویسنده ، کتاب ، تخفیف ، تاریخ ، عشق ، موراکامی ، رئالیسم ، چاپ ، انتشارات بیدگل

خوشه دهم : تخریب ، مرمت ، فرهنگ ، صنایع دستی ، سه تار ، کنسرت ، آواز ، شجریان ، پیانو

همانطور که مشاهده می شود هر خوشه را می توان به یک نوع تیتر خبری مربوط دانست ، مثلا :

خوشه اول : ورزشی / خوشه دوم : قضایی / خوشه سوم : اقتصادی / خوشه چهارم :سیاست خارجی / خوشه پنجم : سیاست داخلی / خوشه ششم : اجتماعی / خوشه هفتم : نظامی / خوشه هشتم : صنعتی / خوشه نهم :کتاب / خوشه دهم : میراث فرهنگی و موسیقی

** لازم به ذکر است در اینجا میتوان برای خوشه بندی دقیق تر تعداد خوشه ها را بیشتر از ۱۰ در نظر گرفت ، به طور مثال می توان خوشه دهم را به دو خوشه جدا (میراث فرهنگی و موسیقی) خوشه بندی کرد و یا حتی با افزایش تعداد خوشه ها می توان مساله multi lable بودن خبر ها (مثلا یک خبر هم اجتماعی باشد هم سیاسی هم اقتصادی) را تا حدودی حل کرد (مثلا خوشه جدیدی داشت با عنوان سیاسی – اجتماعی یا خوشه جدید داشت با عنوان سیاسی – اقتصادی) ، البته برای این تیپ مسایل می توان از خوشه بندی فازی هم استفاده نمود .

برگردیم به ادامه حل مسئله ، تا به اینجای کار توانستیم خبر های مشابه با هم را در خوشه هایی جدا کنیم ، و با توجه به ویژگی های مشترکِ داده های هر خوشه توانستیم نوع داده ی هر خوشه را حدس بزنیم (ورزشی یا قضایی یا اقتصادی یا ...) حال چنانچه قصد تشخیص تیتر خبر برای داده های جدید را با استفاده از داده های فعلی را داشته باشیم ؛ می توانیم تیتر خبری که با استفاده از خوشه بندی برای داده های فعلی به دست آورده بودیم به عنوان برچسپ (lable) برای دادگان فعلی به کار ببریم و مسئله را از unsupervised به یک مسئله supervised تبدیل کرده و از دادگان برچسپ دار فعلی برای آموزش مدلی که بتواند تیتر خبری را برای هر محتوای خبری تشخیص دهد ، استفاده کرد .

منبع :

۱- کتاب درسی

۲ - News Article Clustering Using Unsupervised Learning | by Austin L.E. Krause | Medium

Problem 3.a

ابتدا pytorch را با استفاده از دستور العمل کتاب نصب میکنیم و سپس import می کنیم .

حال متغیر های X و Y را مطابق کتاب تعریف میکنیم :

```
[5]  X = torch.arange(12, dtype=torch.float32).reshape((3,4))
     Y = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
```

```
X
```

```
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.]])
```

```
[7]  Y
```

```
tensor([[2., 1., 4., 3.],
        [1., 2., 3., 4.],
        [4., 3., 2., 1.]])
```

حاصل دو مقایسه خواسته شده در صورت سوال به شکل زیر است :

```
[8]  X<Y
```

```
tensor([[ True, False,  True, False],
        [False, False, False, False],
        [False, False, False, False]])
```

در واقع در اینجا درایه ها نظیر به نظیر مقایسه می شوند و چنانچه درایه موجود در X از درایه موجود در Y کمتر باشد مقدار True و در غیر این صورت مقدار False بر میگرداند .

```
[9]  X>Y

     tensor([[False, False, False, False],
             [ True,  True,  True,  True],
             [ True,  True,  True,  True]])
```

در اینجا هم چنانچه درایه موجود در X از درایه موجود در Y بیشتر باشد مقدار True و در غیر این صورت مقدار False بر میگرداند .

سپس در ادامه سوال می بایست تنسر سه بعدی تعریف کنیم و عمل جمع را بر آن تست کنیم

دو تنسور a1 , b1 را با استفاده از تابع توزیع استاندارد (توزیع گوسی با میانگین 0 و انحراف معیار 1) تعریف میکنیم .

```
[105] a1 = torch.randn(2, 3, 3)
      b1 = torch.randn(2, 3, 3)

      print("a1 is : " ,a1)

      a1 is :  tensor([[[ 0.3682,  1.2486, -0.8974],
                       [ 0.5905, -0.6181,  0.1003],
                       [-0.1786,  0.1607,  0.3776]],

                      [[ 0.1538,  0.3202,  1.6721],
                       [-0.1790, -0.3078,  1.9797],
                       [ 0.5348, -0.8462,  0.1426]]])

[107] print("b1 is : " ,b1)

      b1 is :  tensor([[[ 0.9134, -1.9298,  0.3584],
                       [ 0.0765,  1.0331, -2.7999],
                       [ 1.4186,  1.1320,  0.0271]],

                      [[-0.7812, -0.8303, -2.1633],
                       [ 0.4146, -0.9341,  0.0194],
                       [-1.4283, -0.3302, -0.0236]]])
```

مشاهده می شود چنانچه ابعاد دو تنسور با هم برابر باشد ، خروجی جمع نظیر به نظیر درایه ها می باشد .

```
a1 + b1
```

```
tensor([[[ 1.2816, -0.6811, -0.5390],
         [ 0.6670,  0.4149, -2.6995],
         [ 1.2400,  1.2927,  0.4047]],

        [[-0.6274, -0.5101, -0.4912],
         [ 0.2356, -1.2419,  1.9991],
         [-0.8935, -1.1764,  0.1190]]])
```

حال حالتی را در نظر بگیرید که ابعاد تنسور ها کاملا با هم برابر نباشد و در حداقل یکی از ابعاد با هم تفاوت داشته باشند ؛ در اینجا ممکن است ۲ اتفاق رخ بدهد .

الف : چنانچه اندازه بعد هیچکدام از تنسور ها در بعد یا بعد هایی که با هم تفاوت اندازه دارند برابر ۱ نباشد . اینجا هم a2, b2 را مطابق قسمت قبل و با توزیع نرمال تعریف می کنیم . همانطور که مشاهده میکنید در بعد دوم مقادیر دو تنسور متفاوت است.

```
[109] a2 = torch.randn(2, 4, 3)
      b2 = torch.randn(2, 3, 3)
```

```
[110] print("a2 is : " ,a2)
```

```
a2 is :  tensor([[[-1.1271,  0.4227,  0.7973],
         [ 0.8409, -0.7177, -1.0686],
         [-1.5106,  1.4640, -2.3686],
         [-0.1523, -0.3926,  1.1293]],

        [[ 1.2625,  1.5082, -0.0754],
         [-0.5544, -0.8997,  1.4911],
         [ 0.5425, -0.3502, -1.6006],
         [ 1.3745, -1.2976,  0.2387]]])
```

```
print("b2 is : " ,b2)
```

```
b2 is :  tensor([[[ 1.6258, -0.1811, -0.4063],
         [ 0.7607, -1.8106,  0.1626],
         [-0.3798,  0.4725, -1.2554]],

        [[-0.5511, -0.7853, -0.4899],
         [-0.2179, -0.2276, -1.3092],
         [ 0.3244, -1.4059, -0.6300]]])
```

در صورت اجرا کردن دستور جمع ، با ارور زیر مواجه می شویم که لزوم برابر بودن ابعاد را به ما یاد آوری میکند .

```
a2 + b2
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-113-7294da579f21> in <module>()
----> 1 a2 + b2

RuntimeError: The size of tensor a (4) must match the size of tensor b (3) at non-singleton dimension 1
```

برای تفاوت در بقیه ابعاد هم همین اتفاق می افتد :

```
[116] a3 = torch.randn(3, 3, 3)
      b3 = torch.randn(2, 3, 3)
```

```
[117] a3 +  b3
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-117-8d0c07d0ace0> in <module>()
----> 1 a3 +  b3

RuntimeError: The size of tensor a (3) must match the size of tensor b (2) at non-singleton dimension 0
```

اختلاف در بعد اول

```
a4 = torch.randn(2, 3, 6)
b4 = torch.randn(2, 3, 3)
```

```
a4 + b4
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-119-be99e905e017> in <module>()
----> 1 a4 + b4

RuntimeError: The size of tensor a (6) must match the size of tensor b (3) at non-singleton dimension 2
```

اختلاف در بعد سوم

ب : چنانچه اندازه بعد یکی از تنسور ها در بعد یا بعد هایی که با هم تفاوت انداره دارن ۱ باشد . a5 , b5 را همانند قبل با استفاده از تابع توزیع نرمال تعریف میکنیم به طوری که در بعد اول و سوم تفاوت اندازه داشته باشند:

```
a5 = torch.randn(1, 3, 1)
b5 = torch.randn(2, 3, 3)
```

```
print("a5 is :" , a5)
print("b5 is : " , b5)
```

```
a5 is : tensor([[[ 0.3255],
         [-1.0365],
         [ 0.7870]]])
b5 is :  tensor([[[-0.3455, -0.2843,  0.8249],
         [ 1.6197,  1.5869,  1.0632],
         [-0.3041,  0.1351, -2.0956]],

        [[-0.9203,  0.3144,  0.1861],
         [ 0.3073,  0.0696, -1.3231],
         [ 0.4849,  0.7784, -0.7234]]])
```

حال حاصل جمع را حساب می کنیم :

```
a5 + b5
```

```
tensor([[[-0.0200,  0.0412,  1.1504],
         [ 0.5831,  0.5503,  0.0266],
         [ 0.4829,  0.9221, -1.3085]],

        [[-0.5948,  0.6399,  0.5116],
         [-0.7292, -0.9669, -2.3597],
         [ 1.2719,  1.5655,  0.0637]]])
```

مشاهده می شود که عملیات بدون خطا انجام می شود و در واقع a5 را به صورت زیر حساب کرده و سپس عملیات جمع نظیر به نظیر انجام شده است

```
tensor([[[ 0.3255,  0.3255,  0.3255],
         [-1.0365, -1.0365, -1.0365],
         [ 0.7870,  0.7870,  0.7870]],

        [[ 0.3255,  0.3255,  0.3255],
         [-1.0365, -1.0365, -1.0365],
         [ 0.7870,  0.7870,  0.7870]]])
```

در واقع a5 از انجا که اندازه بعدش در آن بعدی که با b5 تفاوت داشته و برابر ۱ بوده ، برنامه ابعادش را به ابعاد مساوری با ابعاد b5 قرار داده و درایه های جدید را مطابق درایه های قبلی پر می کند .

نتیجه : عملگر ها بر تنسورها در حالتی بدون خطا کار می کنند که یا ابعاد تنسور ها کاملا با هم برابر باشد ، یا اگرهم فرق دارد ، اندازه بعدی که فرق دارد مقدار ۱ باشد ، در غیر این صورت با خطا مواجه خواهیم شد .

کد این قسمت از سوال در فایل ضمیمه (.ipynb) و در قسمت Problem 3.a قرارا داده شده است .

Problem 3.b

بعد از فراخوانی کردن کتابخانه های مورد نیاز ، داده ی مورد نیاز را به صورت dictionary ایجاد می کنیم .

```
import pandas as pd
import tensorflow as tf
```

```
[94] dataset = {
        'column1':[0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 ],
        'column2':[0 , 2 , 4 , 6 , 8 , 10 , 12 , 14 , 16 , 18 ],
        'column3':  pd.Series([0 , 3 , 6 , 9 , 12 , 15] , index = [0 , 1 , 2 , 7 , 8 ,9]),
        'column4':  pd.Series([0 , 3 , 4 , 12 , 2, 3 ] , index = [0 , 1 , 2 , 7 , 8 ,9]),
        'column5':  pd.Series([1 , 1 , 1, 1, 1 ,1] , index = [0 , 1 , 2 , 7 , 8 ,9]) ,
        'column6':  pd.Series([3 ,3 ] , index = [0 , 1 ]),
        'column7':  pd.Series([4 ,4 ,4] , index = [0 , 1 , 2 ]),
        'column8':  pd.Series([10 ,10] , index = [0 , 1]),
    }
```

سپس آن را به فرمت دیتا فریم به جهت حذف ستون با بیشترین مقدار NaN تبدیل میکنیم

```
dataframe = pd.DataFrame(dataset, index = [0,1,2,3,4,5,6,7,8,9] )
print(dataframe)
```

	column1	column2	column3	column4	column5	column6	column7	column8
0	0	0	0.0	0.0	1.0	3.0	4.0	10.0
1	1	2	3.0	3.0	1.0	3.0	4.0	10.0
2	2	4	6.0	4.0	1.0	NaN	4.0	NaN
3	3	6	NaN	NaN	NaN	NaN	NaN	NaN
4	4	8	NaN	NaN	NaN	NaN	NaN	NaN
5	5	10	NaN	NaN	NaN	NaN	NaN	NaN
6	6	12	NaN	NaN	NaN	NaN	NaN	NaN
7	7	14	9.0	12.0	1.0	NaN	NaN	NaN
8	8	16	12.0	2.0	1.0	NaN	NaN	NaN
9	9	18	15.0	3.0	1.0	NaN	NaN	NaN

در ادامه طبق کد زیر ، در حلقه اول مقدار بیشترین NaN موجود در یک ستون را یافته و در متغیر flag ذخیره می کنیم.

و در حلقه دوم نام ستون هایی که مقدار NaN در آن ها برابر با flag به دست آمده از حلقه اول است را به صورت لیست در متغیرلیستی index ذخیره می کنیم.

مشاهده می شود که بیشترین مقدار NaN در یک ستون برابر ۸ است ، در ستون های ششم و هشتم

```
flag=0
index =[]
for i in dataframe.columns:
  if sum(dataframe[i].isna()) >= flag :
    flag = sum(dataframe[i].isna())
for i in dataframe.columns:
  if sum(dataframe[i].isna()) == flag :
    index.append(i)
print(index)
print(flag)
```

```
['column6', 'column8']
8
```

حال با استفاده از دستور drop. ستون های موجود در متغیر index (ششم و هشتم) را حذف می کنیم

```
[99] dataframe = dataframe.drop(index , axis =1 )
```

```
dataframe
```

	column1	column2	column3	column4	column5	column7
0	0	0	0.0	0.0	1.0	4.0
1	1	2	3.0	3.0	1.0	4.0
2	2	4	6.0	4.0	1.0	4.0
3	3	6	NaN	NaN	NaN	NaN
4	4	8	NaN	NaN	NaN	NaN
5	5	10	NaN	NaN	NaN	NaN
6	6	12	NaN	NaN	NaN	NaN
7	7	14	9.0	12.0	1.0	NaN
8	8	16	12.0	2.0	1.0	NaN
9	9	18	15.0	3.0	1.0	NaN

سپس با استفاده از شبه کد زیر ، داده نهایی را به فرمت تنسور تبدیل میکنیم

```
tensorflow1 = tf.convert_to_tensor(dataframe)
```

```
tensorflow1
```

```
<tf.Tensor: shape=(10, 6), dtype=float64, numpy=
array([[ 0.,  0.,  0.,  0.,  1.,  4.],
       [ 1.,  2.,  3.,  3.,  1.,  4.],
       [ 2.,  4.,  6.,  4.,  1.,  4.],
       [ 3.,  6., nan, nan, nan, nan],
       [ 4.,  8., nan, nan, nan, nan],
       [ 5., 10., nan, nan, nan, nan],
       [ 6., 12., nan, nan, nan, nan],
       [ 7., 14.,  9., 12.,  1., nan],
       [ 8., 16., 12.,  2.,  1., nan],
       [ 9., 18., 15.,  3.,  1., nan]])>
```

کد این قسمت از سوال در فایل ضمیمه (.ipynb) و در قسمت Problem 3.b قرارا داده شده است .