دانشگاه صنعتی اصفهان

مرکز تحقیقاتی هوش مصنوعی

گزارش کارآموزی

موضوع کارآموزی: Deep Time-Series Clustering

ترم تحصیلی: تابستان ۱۴۰۱

استاد کارآموزی: دکتر سید جلال ذهبی

نویسنده گزارش: علیرضا حبیبی

۱ مقدمه

Time-series یا سری های زمانی دنباله ای از داده ها هستند که در طول زمان محدودی جمع آوری شده و بر اساس زمان چیده شده اند. بر اساس [۱] سری های زمانی را میتوان به چهار دسته تقسیم کرد:

1- Univariate

2- Multivariate

3- Tensor Fields

4- Multifield

تمزکز ما در این دوره کارآموزی بر روی سری های زمانی Multvariate یا سری های زمانی چند متغیره است.

در دسته بندی سری های زمانی چالش هایی از جمله زیاد بودن و وابستگی زیاد ویژگی ها، وجود نویز، و غیره وجود دارد، از این رو روش های دسته بندی عمیق برای این عمل پیشنهاد شده اند و به این دلیل که برچسب زدن مغادیر بزرگ از سری های زمانی عملی دشوار است و این دست داده ها کمیاب هستند، باید از الگوریتم های Unsupervised Learning برای این عمل استفاده کرد.

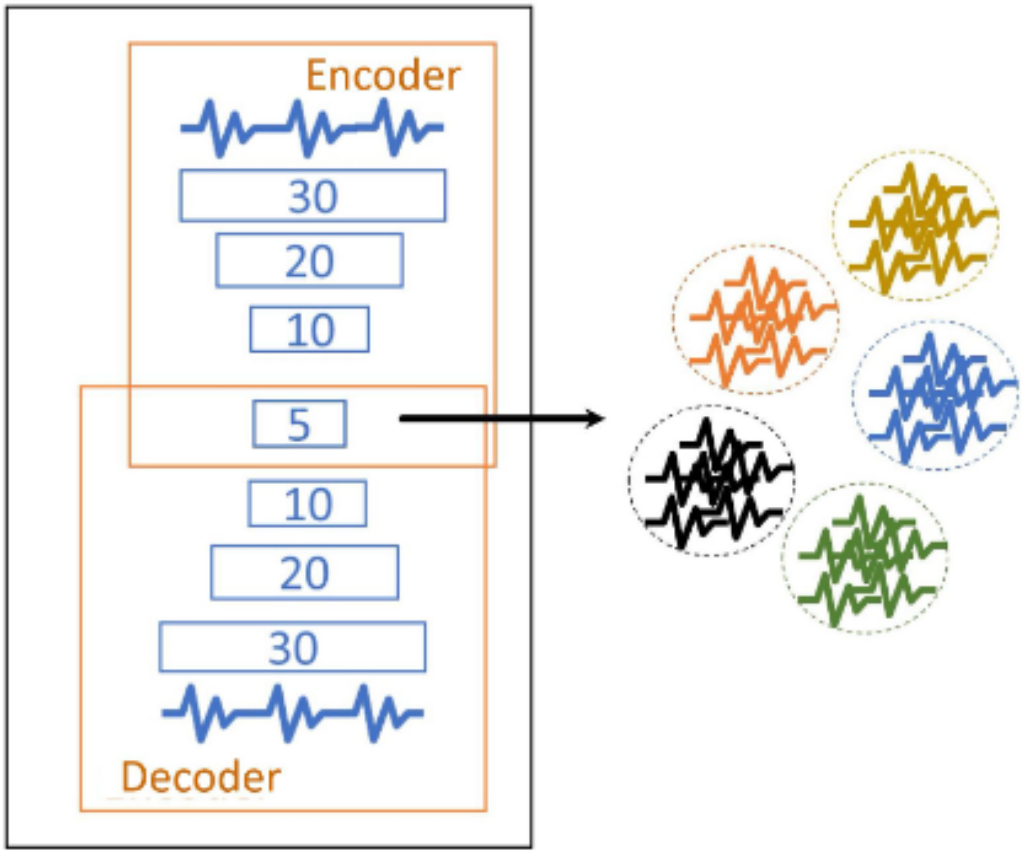
۲ Deep Time-Series Clustering

روش کلی مدلسازی ما در این دوره کارآموزی طبق [۱] میباشد. مدل پیشنهادی در این روش شامل یک Autoencode است که از آن برای استخراج ویژگی استفاده میشود. استخراج ویژگی در دسته بندی سری های زمانی بسیار مهم است زیرا همانطور که قبلا اشاره شد تعداد ویژگی ها و وابستگی آنها زیاد است پس باید تعداد مشخصی ویژگی استخراج شود و این تعداد باید نماینده کل سری زمانی باشند و در عین حال تعداد آنها در حدی باشد که زمان اجرای الگوریتم دسته بندی بهینه باشد.

در روش اشاره شده، بعد از استخراج ویژگی ها باید الگوریتم دسته بندی مورد نظر را روی ویژگی های استخراج شده اجرا کرد و سپس با استفاده از Autoencode داده های اولیه با تعداد ویژگی بالا را بازسازی کرد. در منبع مورد نظر چندین پیاده سازی مختلف پیشنهاد شده اند که از بین آنها روش پیشنهادی در [۲] را برای پیاده سازی انتخاب کردیم.

۱.۲ Autoencoder

همانطور که قبلا اشاره شد اتوانکدر ها برای استخراج ویژگی ها و کاهش ابعاد داده استفاده میشوند. ساختمان کلی اتوانکدرها مانند شکل ۱ است.



شکل ۱: ساختمان کلی اتوانکدرها - منبع:[۱]

این اتوانکدر را طبق تصاویر زیر پیاده سازی کردیم.

```python
class DAE(nn.Module):

    def __init__(self):
        super(DAE, self).__init__()

        self.fc1 = nn.Linear(13, 30)
        self.fc2 = nn.Linear(30, 20)
        self.fc3 = nn.Linear(20, 10)
        self.fc4 = nn.Linear(10, 6)

        self.fcr4 = nn.Linear(30, 13)
        self.fcr3 = nn.Linear(20, 30)
        self.fcr2 = nn.Linear(10, 20)
        self.fcr1 = nn.Linear(6, 10)
        self.droput = nn.Dropout(0.2)



    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))

        x = F.relu(self.fcr1(x))
        x = F.relu(self.fcr2(x))
        x = F.relu(self.fcr3(x))
        x = (self.fcr4(x))
        return (x)
```

شکل ۲: ساخت مدل

```
[12] for epoch in range(500):
       y_pred=model(xtrain)
       l=loss(y_pred,ytrain)
       optimizer.zero_grad()
       l.backward()
       optimizer.step()
       if epoch%50==0:
         print(l)
```

```
tensor(0.9354, grad_fn=<MseLossBackward0>)
tensor(0.6834, grad_fn=<MseLossBackward0>)
tensor(0.5037, grad_fn=<MseLossBackward0>)
tensor(0.4195, grad_fn=<MseLossBackward0>)
tensor(0.3781, grad_fn=<MseLossBackward0>)
tensor(0.3529, grad_fn=<MseLossBackward0>)
tensor(0.3382, grad_fn=<MseLossBackward0>)
tensor(0.3265, grad_fn=<MseLossBackward0>)
tensor(0.3178, grad_fn=<MseLossBackward0>)
tensor(0.3104, grad_fn=<MseLossBackward0>)
```

```
print(model(xtest[5]))
print(xtest[5])
```

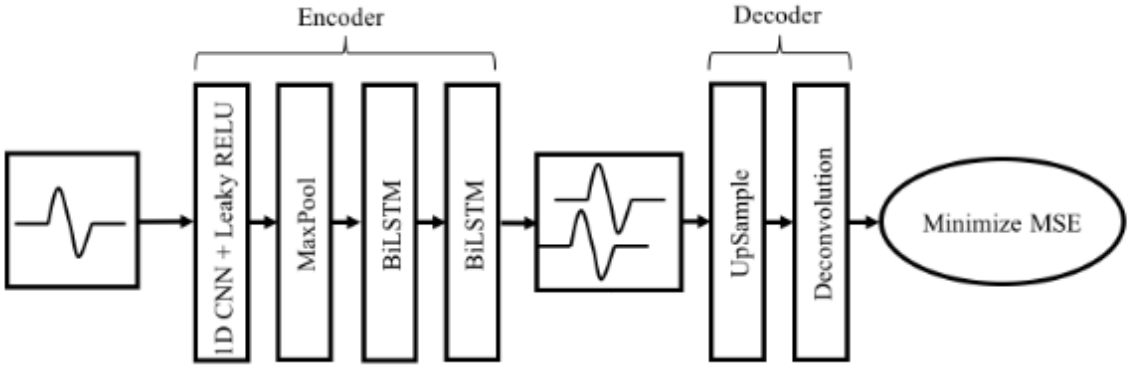
```
tensor([1.2687, 1.0776, 1.0025, 0.8840, 1.0105, 1.0794, 0.9130, 0.8566, 1.2304,
        1.0096, 0.9372, 0.9465, 0.9461], grad_fn=<AddBackward0>)
tensor([1.2850e+01, 1.6000e+00, 2.5200e+00, 1.7800e+01, 9.5000e+01, 2.4800e+00,
        2.3700e+00, 2.6000e-01, 1.4600e+00, 3.9300e+00, 1.0900e+00, 3.6300e+00,
        1.0150e+03])
```

شکل ۳: آموزش مدل

مدل های اتوانکدر دارای فضایی به نام latent space هستند که در واقع همان فضای بین انکدر و دیکدر است که در آن ویژگی های داده ورودی استخراج شده اند(در شکل ۱ پنج ویژگی استخراج شده است).

مدلهای اتوانکدر میتوانند شامل لایه های کانولوشنی (convolutional) یا لایه های فولی کانکتد یا لایه های بازگشتی (recurrent) باشند. مدل مدنظر ما برای قسمت انکدر خود دارای یک لایه کانولوشنی با سایز فیلتز ۱۰، یک لایه max pooling با اندازه فیلتر به صورتی که خروجی کمتر از ۱۰۰ ویژگی داشته باشد و دو لایه بازگشتی از نوع BI-LSTM است و برای قسمت دیکدر دارای یک لایه upsampling با سایز فیلتر مشابه با سایز فیلتز لایه max pooling و یک لایه دیکانولوشن با سایز فیلتر ۱۰ است. برای فعالسازی لایه ها نیز از تابع Leaky Rectifying Linear Units(L-ReLU) استفاده میکنیم.

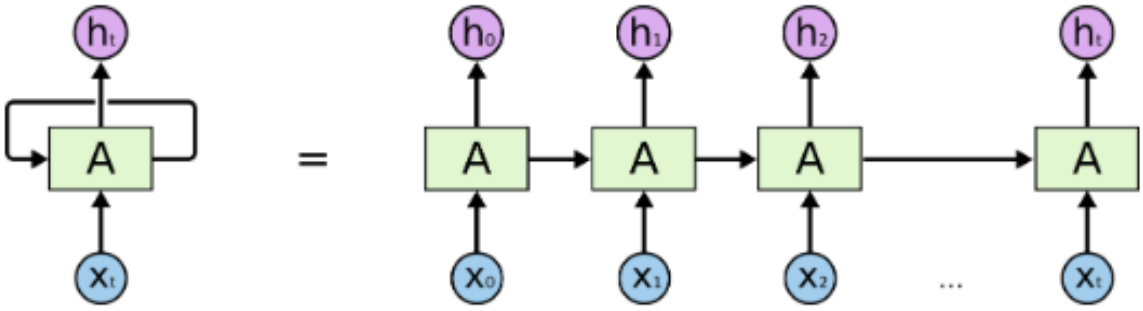
شکل ۴ نمای کلی مدل اتوانکدر موردنظر را نمایان است.



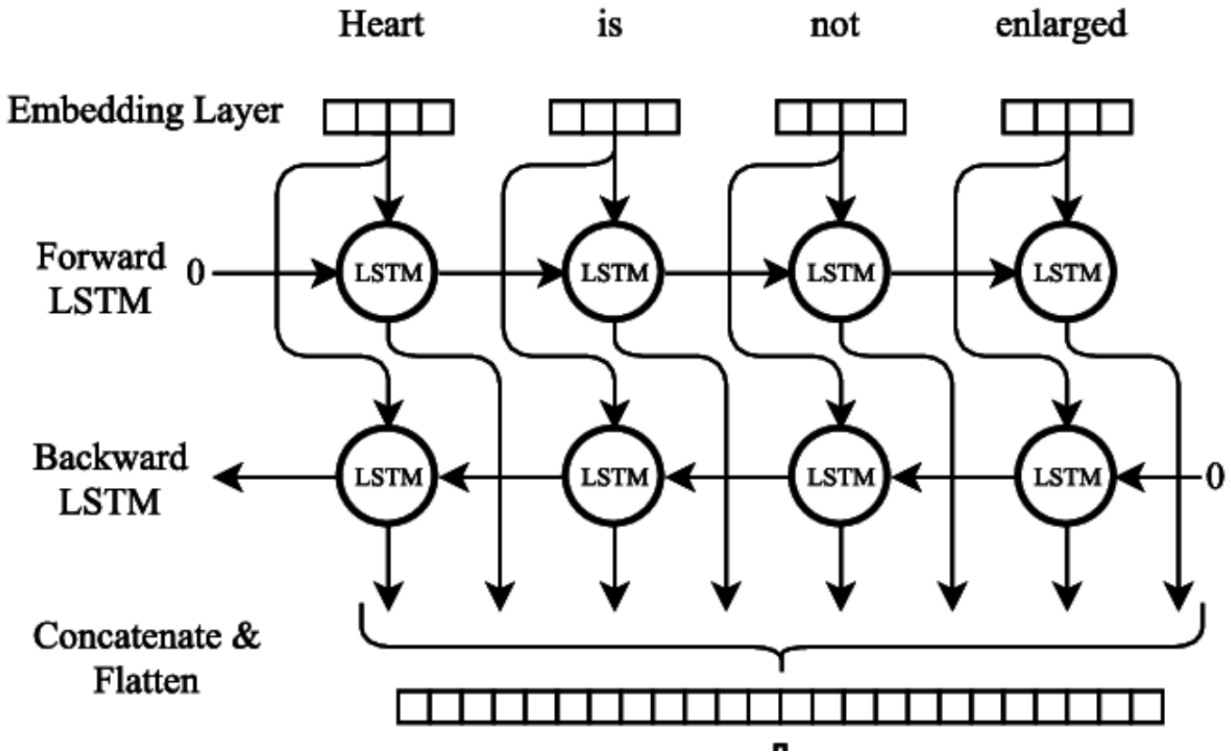
شکل ۴: ساختمان کلی اتورانکدر موردبحث- منبع:[۲]

میتوان گفت ساختار لایه های BI-LSTM شامل دو لایه LSTM .است، یکی در جهت اول به آخر داده و دیگری در جهت آخر به اول. لایه های LSTM لایه هایی هستند که در خود حلقه هایی دارند که باعث میشود تقدم ویژگی های داده نسبت به یکدیگر تاثیر بیشتری در شبکه پیدا کند. این امر به این گونه میسر میشود که خروجی هر نود در یک لایه به ورودی نود کناری خود متصل است.

شبکه های BI-LSTM با دو طرفه کردن این ارتباط، تاثیر تقدم و تاخر در قسمت های مختلف داده را بیشتر میکنند. شکل ۵ نمایانگر لایه بازگشتی ساده و شکل ۶ نمایانگر ساختار لایه های BI-LSTM هستند.



شکل ۵: ساختار LSTM - منبع:https://colah.github.io/posts/2015-08-Understanding-LSTMs/



شکل ۶: ساختار BI-LSTM - منبع:https://paperswithcode.com/method/bilstm

طبق [۲] برای بهینه بودن زمان پردازش، بهتر است کمتر از ۱۰۰ ویژگی توسط اتوانکدر مربوطه استخراج شوند. همینطور باید قبل از آموزش قسمت دسته بندی، قسمت اتوانکدر را آموزش قبلی داد تا قبل از اجرای دسته بندی فضای پنهان (latent space) معناداری داشته باشیم.این اکودر را با Adam optimize و تابع هزینه Mean Squere Error(MSE)، برای ۱۰ ایپاک آموزش قبلی میدهیم. باید به این نکته دقت کرد که هنگام آموزش قسمت دسته بندی، قسمت اتوانکدر را نیز همزمان آموزش دوباره میدهیم.

۳ Clustering and DTC Model

در این مدل برای دسته بندی، به تعداد مورد نظر خوشه داریم که برای مقداردهی مراکز این خوشه ها، الگوریتم Hierarchical Clustering را بر روی داده های مورد نظر و پس از استخراج ویژگی ها اجرا میکنیم. نحوه کلی عملکرد این الگوریتم به این صورت است که در ابتدا فرض میکنیم به تعداد داده ها خوشه داریم و هر داده مرکز یکی از خوشه ها است. سپس جفت خوشه هایی که به یکدیگر نزدیک هستند را انتخاب میکنیم و هر جفت را تبدیل به یک خوشه میکنیم و مرکز خوشه را میانگین مراکز اولیه دو خوشه قرار میدهیم. این عمل را مرتب تکرار میکنیم تا به تعداد خوشه موردنظر برسیم.

پس از مقداردهی اولیه مراکز خوشه ها با استفاده از الگوریتمی غیرنظارتی استفاده میکنیم که دو مرحله دارد: ۱-محاسبه احتمال

تعلق هر داده(پس از استخراج ویژگی ها) به هر یک از خوشه ها ۲-به روزرسانی مراکز خوشه ها با توجه به تابع هزینه مورد نظر برای محاسبه این دو مرحله، با توجه به روابط زیر عمل میکنیم.

$$q_{ij} = \frac{(1 + \sqrt{z_i^2 + w_j^2})^{-}1}{\sum_{j=1}^{k}(1 + \sqrt{z_i^2 + w_j^2})^{-}1}$$

$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j=1}^{k} q_{ij}^2 / f_j}$$

$$f_j = \sum_{i=1}^{n} q_{ij}$$

$$L = \sum_{i=1}^{n}\sum_{j=1}^{k} p_{ij} log \frac{p_{ij}}{q_{ij}}$$

q_{ij} نشان دهنده احتمال تعلق داده iام به خوشه jام است. z_i نشاندهنده داده iام است و w_j نشان دهنده خوشه jام است و k تعداد کل خوشه ها است. سپس با استفاده از الگوریتم Gradient Descent و مشتق زیر خوشه ها را بروزرسانی میکنیم.

$$\frac{dL_c}{dw_i} = \frac{1}{2}\sum_{j}(1 + \sqrt{z_i^2 + w_j^2}) * (p_{ij} - q_{ij}) \frac{d(\sqrt{z_i^2 + w_j^2})}{dw_i}$$

با learning rate = 0.1 مدل را تا رسیدن به همگرایی آموزش میدهیم.

۴ نتیجه

ما در این دوره کارآموزی مدل پیاده سازی شده را با دیتاست Human Activity Recognition(HAR) آموزش دادیم. این دیتاست شامل اطلاعات ذخیره شده توسط تلفن همراه هوشمند در دست سی نفر در شش حال ایستادن، نشستن، درازکشیدن، راه رفتن، بالا رفتن از پله و پایین آمدن از پله است. دیتاست مورد استفاده شامل تقریبا هفت هزار داده بود که هرکدام پانصدوشصت و یک ویژگی داشت. قسمت های مختلف پیاده سازی در شکل های زیر آمده است.

```
[ ]  class DTC(nn.Module):

        def __init__(self,maxpooling_filter):
            super(DTC, self).__init__()


            self.cnv1=nn.Conv1d(1, 1, 10)
            self.fc1=nn.Linear(552-maxpooling_filter+1,552-maxpooling_filter+1)
            self.fc2=nn.Linear(552-maxpooling_filter+1,552-maxpooling_filter+1)
            self.lrel=nn.LeakyReLU()
            self.maxpool=nn.MaxPool1d(maxpooling_filter, stride=1,return_indices=True)
            self.maxunpool=nn.MaxUnpool1d(maxpooling_filter, stride=1)
            self.upsample=nn.Upsample(scale_factor=(552/(553-maxpooling_filter)), mode='bilinear')
            self.deconv1=nn.ConvTranspose1d(1, 1, 10)

        def encode(self, x):
            x = self.lrel(self.cnv1(x))

            x,self.indices= self.maxpool(x)
            x = self.lrel(self.fc1(x))
            return x,self.indices
        def decode(self,x,indice):
            x = self.lrel(self.fc2(x))
            x= (self.maxunpool(x,indice))#.squeeze(dim=3)

            x= self.deconv1(x)
            return x
```

شکل ۷: کد مدل

```
model.train()
for epoch in range(50):
  for features,labels in data:
    optimizer.zero_grad()
    x_pred,indice=model.encode(features)

    x_pred=model.decode(x_pred,indice)

    l=loss(x_pred,features)

    l.backward()
    optimizer.step()
  print(l)
```

```
tensor(0.8287, grad_fn=<MseLossBackward0>)
tensor(0.8029, grad_fn=<MseLossBackward0>)
tensor(0.7513, grad_fn=<MseLossBackward0>)
tensor(0.7505, grad_fn=<MseLossBackward0>)
tensor(0.7428, grad_fn=<MseLossBackward0>)
tensor(0.7181, grad_fn=<MseLossBackward0>)
tensor(0.6831, grad_fn=<MseLossBackward0>)
tensor(0.6621, grad_fn=<MseLossBackward0>)
```

شکل ۸: کد آموزش اولیه اتوانکدر

```
n_clusters=6
for epoch in range(50):
  for features,labels in data:
    optimizer2.zero_grad()
    model2 = AgglomerativeClustering(
        n_clusters=n_clusters, linkage="complete"
    )
    with torch.no_grad():
      x_pred,indice=model.encode(features)
      a,b,c=features.shape
      x_pred=x_pred.view(a,33)
      y_predict=model2.fit_predict(x_pred)
      clf = NearestCentroid()
      clf.fit(x_pred, y_predict)
    qij = torch.ones([batch_size, n_clusters], dtype=torch.float64, requires_grad=True)
    pij = torch.ones([batch_size, n_clusters], dtype=torch.float64, requires_grad=True)
    fj = torch.ones([n_clusters,1], dtype=torch.float64, requires_grad=True)
    sigma1=0
    for i , feature in enumerate(features):
      sigma1=0
      for j in range(n_clusters):
        sigma1+=((1+torch.sqrt(torch.sum((x_pred[i]-clf.centroids_[j])**2)))**-1)
      for j in range(n_clusters):
        qij[i][j] = (((1+torch.sqrt(torch.sum((x_pred[i]-clf.centroids_[j])**2)))**-1)/sigma1)
    for j in range(n_clusters):
      for i , feature in enumerate(features):
        print (qij.shape)
        fj[j]+=qij[i][j]
    sigma1=0
    cluster_loss=0
    for i , feature in enumerate(features):
      for j in range(n_clusters):
        cluster_loss+=pij[i][j]*torch.log(pij[i][j]/qij[i][j])
    cluster_loss.backward()
    optimizer2.step()
```

شکل ۹: کد احتساب مقدار اولیه مرکز خوشه ها و تابع هزینه الگوریتم دسته بندی

مراجع

[۱] Deep Jones، W. M. and Xie، X. Ali، M. Alqahtani، A. A clustering: time-series Deep، Electronics review، vol. ۱۰، no. ۲۳، p. ۳۰۰۱، ۲۰۲۱.

[۲] Deep Madiraju، S. N. Fully clustering: temporal Deep، features. time-domain of learning unsupervised، Ph.D. dissertation، State Arizona University، ۲۰۱۸.