به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین چهارم

پرسش ۱	نام و نام خانوادگی	آرمین قاسمی
	شماره دانشجویی	۸۱۰۱۰۰۱۹۸
پرسش ۲	نام و نام خانوادگی	امیرحسین ثمودی
	شماره دانشجویی	۸۱۰۱۰۰۱۰۸
	مهلت ارسال پاسخ	۱۴۰۳.۱۲.۲۷

فهرست

شکل‌ها

جدول‌ها

پرسش ۱ – توصیف تصویر با شبکه ترکیبی ResNet50 + LSTM-GRU

۱-۱: مقدمه

در این تمرین قصد داریم که یک شبکه عمیق برای تولید متن برای تصویر ورودی را توسعه دهیم. این شاخه که Image-Captioning نام دارد معمولا از شبکه هایی استفاده میکند که در ابتدا یک شبکه عمیق CNN ویژگی های تصویر را استخراج میکند (بخش Encoder)، و سپس با کمک شبکه های بازگشتی (RNN) این ویژگی ها را به جملات روان تبدیل می‌نماید. (بخش Decoder)

در این سوال بر روی تصاویر مجموعه داده Flickr8k کپشن ها را تولید می‌کنیم. همچنین برای بخش Decoder، از یک معماری ترکیبی که در مقاله معرفی شده است استفاده می نماییم.

۱-۲: مجموعه داده و پیش‌پردازش

۱: انتخاب مجموعه داده

ابتدا مجموعه داده Flickr8k را دانلود میکنیم. این مجموعه داده شامل حدودا ۸۰۰۰ تصویر و یک فایل captions.txt می‌باشد که برای هر تصویر، ۵ کپشن مختلف را شامل می‌شود.

captions.txt

```
image,caption
1000268201_693b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg,A girl going into a wooden building .
1000268201_693b08cb0e.jpg,A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg,A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg,A little girl in a pink dress going into a wooden cabin .
1001773457_577c3a7d70.jpg,A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg,A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg,A black dog and a white dog with brown spots are staring at each other in the street .
1001773457_577c3a7d70.jpg,Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg,Two dogs on pavement moving toward each other .
```

شکل ۱: بخش از فایل کپشن های تصاویر

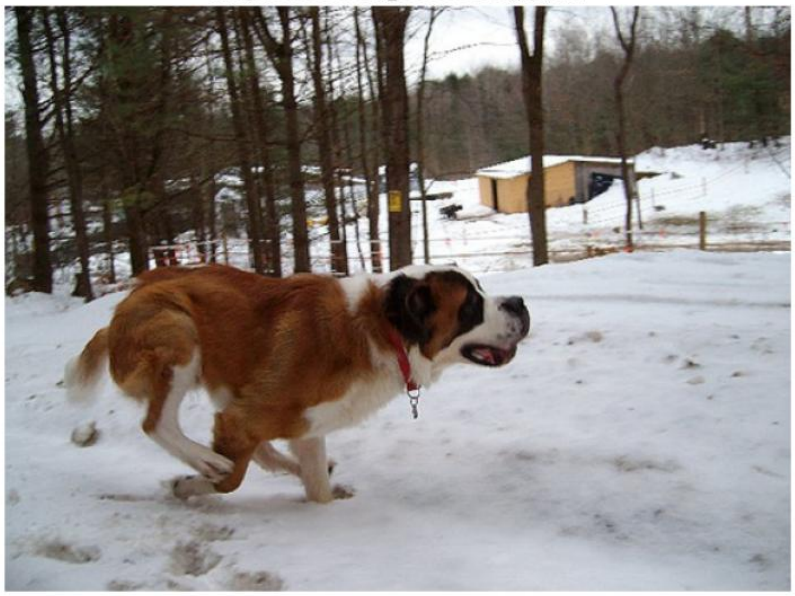
حال تابعی مینویسیم که فایل کپشن ها را بخواند و برای تعدادی از تصاویر، کپشن های متناظرشان را نشان دهد.

Image: 216172386_9ac5356dae.jpg (1/5)

```
1. A bike sits atop a rise with mountains in the background .
2. A man wearing a red uniform and helmet stands on his motorbike .
3. A motocross bike is being ridden over rocks .
4. A motocross biker about to descend
5. The motorcyclist has reached the summit .
```

شکل ۲: نمونه تصویر اول همراه با کپشن

Image: 2135502491_a15c6b5eae.jpg (2/5)



```
1. A beautiful brown and white St Bernard running in the snow
2. A St Bernard lunges through the snow .
3. Large St Bernard dog wearing red collar galloping through the snow .
4. St Bernard dog running in the snowy field .
5. The large brown and white dog is running through the snow .
```

شکل ۴: نمونه تصویر دوم همراه با کپشن

شکل ۵ شکل نمونه سوم همراه با کپشن

۲: پیش‌پردازش تصاویر

در این مرحله تصاویر ورودی را پردازش می‌کنیم. تصاویر را به اندازه ۲۲۴ در ۲۲۴ در می آوریم تا با ابعاد ورودی شبکه ResNet50 سازگار باشند. همچنین برای نرمال سازی تصاویر حول میانگین صفر و انحراف معیار یک، با توجه به اینکه از شبکه ResNet50 برای استخراج ویژگی ها استفاده می کنیم و این شبکه بر روی دیتا ست ImageNet تمرین داده شده است، از مقادیر میانگین و انحراف معیار این دیتا ست (در ۳ کانال مختلف) استفاده می کنیم. این کار باعث سازگار شدن تصاویر با مدل ResNet50 می شود زیرا با داده هایی با این ویژگی آموزش دیده است و در صورت عدم اعمال این نرمال سازی، شبکه CNN می تواند دچار افت دقت قابل توجهی شود.

```
img_size = 224
imagenet_mean = [0.485, 0.456, 0.406]
imagenet_std = [0.229, 0.224, 0.225]

image_transform = transforms.Compose([
    transforms.Resize((img_size, img_size)),
    transforms.ToTensor(),
    transforms.Normalize(mean=imagenet_mean, std=imagenet_std)
])
```

شکل ۶: نرمال سازی تصاویر ورودی

۳: پیش پردازش متن (کپشن‌ها)

حال می‌خواهیم کپشن های تصاویر را پردازش کنیم و یک دیکشنری از کلمات استفاده شده در آن ها برای آموزش مدل بسازیم.

در مرحله اول، یک پیش پردازش ساده روی متن کپشن ها کرده و همه حروف بزرگ را کوچک کرده و همچنین علائم نگارشی (Punctuatoions)، اعداد و نماد هایی غیر از حرف الفبا را حذف می‌کنیم.

سپس Tokenization را انجام میدهیم:

ابتدا همه کپشن ها را خوانده و پیش پردازش اولیه را روی آن ها انجام میدهیم. سپس همه متن را Tokenize میکنیم (کلماتی که با فاصله از هم جدا شده اند یک توکن در نظر گرفته میشوند). در این مرحله ۴ توکن ویژه که عبارتند از <sos> (شروع جمله)، <eos> (پایان جمله)، <pad> (برای پر کردن فضای خالی) و <unk> (برای کلمات جدید و خارج از دیکشنری) را نیز به لیست توکن ها اضافه میکنیم.

سپس به هر یک از توکن های ایجاد شده یک شناسه عددی یکتا داده و دیکشنری ایجاد شده را در یک فایل JSON ذخیره می کنیم.

vocab.json


```
{
    "<pad>": 0,
    "<sos>": 1,
    "<eos>": 2,
    "<unk>": 3,
    "a": 4,
    "child": 5,
    "in": 6,
    "pink": 7,
    "dress": 8,
    "is": 9,
    "climbing": 10,
    "up": 11,
    "set": 12,
    "of": 13,
    "stairs": 14,
    "an": 15,
    "entry": 16,
    "way": 17,
    "girl": 18,
    "going": 19,
    "into": 20,
    "wooden": 21,
    "building": 22,
    "little": 23,
    "playhouse": 24,
    "the": 25,
    "to": 26,
    "her": 27,
    "cabin": 28,
    "black": 29,
    "dog": 30,
    "and": 31,
```


شکل ۷: بخشی از فایل دیکشنری

تعداد کل توکن های ایجاد شده ۸۷۸۲ می‌باشد.

حال برای ساده کردن محاسبات در بخش Decoder، طول همه کپشن ها را برابر ۲۰ در نظر می‌گیریم.

برای این کار تابعی می نویسیم که کپشن ورودی را Tokenize کرده و سپس طول آن را به ۲۰ برساند. ابتدا کپشن را پیش پردازش کرده و توکنایز میکنید. سپس توکن <sos> را به عنوان اولین توکن قرار میدهد و به انتها نیز توکن <eos> اضافه میکند. حال طول کپشن را بررسی میکنیم، درصورتی که کمتر از طول مد نظر (در اینجا ۲۰) بود، به تعداد لازم توکن <pad> اضافه میکنیم. در شرایطی هم که طول آن بیشتر از ۲۰ شود، با حفظ توکن های شروع و پایان، ۱۸ توکن ابتدایی از کپشن را نگه می دارد.

در نهایت همه کپشن های توکنایز شده متناظر هر عکس (که با شناسه شان مشخص میشوند) را در فایل proessed_captions.JSON ذخیره می‌نماییم.

```
Image: 1000268201_693b08cb0e.jpg
  Caption 1 (IDs): [1, 4, 5, 6, 4, 7, 8, 9, 10, 11, 4, 12, 13, 14, 6, 15, 16, 17, 2, 0]
    Length: 20
    As words: <sos> a child in a pink dress is climbing up a set of stairs in an entry way <eos> <pad>
  Caption 2 (IDs): [1, 4, 18, 19, 20, 4, 21, 22, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    Length: 20
    As words: <sos> a girl going into a wooden building <eos> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
  Caption 3 (IDs): [1, 4, 23, 18, 10, 20, 4, 21, 24, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    Length: 20
    As words: <sos> a little girl climbing into a wooden playhouse <eos> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>

Image: 1001773457_577c3a7d70.jpg
  Caption 1 (IDs): [1, 4, 29, 30, 31, 4, 32, 30, 33, 34, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    Length: 20
    As words: <sos> a black dog and a spotted dog are fighting <eos> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
  Caption 2 (IDs): [1, 4, 29, 30, 31, 4, 35, 30, 36, 37, 38, 39, 40, 25, 41, 2, 0, 0, 0, 0]
    Length: 20
    As words: <sos> a black dog and a tricolored dog playing with each other on the road <eos> <pad> <pad> <pad> <pad>
  Caption 3 (IDs): [1, 4, 29, 30, 31, 4, 42, 30, 37, 43, 44, 33, 45, 46, 38, 39, 6, 25, 47, 2]
    Length: 20
    As words: <sos> a black dog and a white dog with brown spots are staring at each other in the street <eos>
```

شکل ۸: نمونه ای کپشن های پردازش شده

تصویر بالا، چند نمونه از کپشن های پردازش شده یک تصویر را نشان میدهد. برای مثال در تصویر اول میبینیم که کپشن ها چگونه توکنایز شده و توکن های <pad> برای ثابت کردن طول به آنها اضافه شده اند. همچنین برای صحت سنجی، کپشن توکنایز شده را با استفاده از دیکشنری به معادل متنی آن تبدیل کرده ایم.

اعمال Padding و یکسان کردن طول ورودی های مدل از این جهت ضروری است که ما در آموزش مدل هایمان معمولا برای افزایش سرعت محاسبات، داده ها را در قالب Batch هایی در می آوریم که هر کدام شامل تعدادی داده ورودی می باشد و محاسبات را تا حدی موازی سازی میکنیم. ولی در صورتی این امر امکان پذیر است که همه داده های یک Batch سایز برابری داشته باشند تا بتوان از ماتریس های Tensor استفاده نمود.

پس با اعمال Padding و یکسان کردن طول همه ورودی ها، میتوان داده ها را در قالب Batch به شبکه اعمال نمود.

۳: تقسیم داده‌ها

در این بخش داده ها را بر اساس تصاویر به سه دسته آموزش (۸۰ درصد)، اعتبارسنجی (۱۰ درصد) و تست (۱۰ درصد) تقسیم میکنیم.

ابتدا فایل proessed_captions که در بخش قبل ایجاد کردیم را خوانده و داده ها را برای جلوگیری از بایاس شبکه Shuffle میکنیم. سپس آن ها را با نسبت مشخص شده تقسیم کرده و در فایل های جداگانه ذخیره میکنیم. تعداد داده های هر دسته به صورت زیر می باشد:

```
Number of images for training: 6472
Number of images for validation: 809
Number of images for testing: 810
```

حال کلاس زیر را برای لود کردن داده ها (تصاویر و کپشن ها) استفاده می کنیم.

```
class Flickr8kDataset(Dataset):
    def __init__(self, json_file_path: str, images_dir_path: str,
                 vocab,
                 transform=None, fixed_caption_length=None):
        super().__init__()
        with open(json_file_path, 'r') as f:
            self.data = json.load(f)
        self.images_dir_path = images_dir_path
        self.transform = transform
        self.vocab = vocab
        self.samples = []
        for img_filename, captions_list in self.data.items():
            for numerical_caption in captions_list:
                self.samples.append((img_filename, numerical_caption))
        if not self.samples:
            raise ValueError(f"No samples found.")
        if fixed_caption_length is None:
            self.fixed_caption_length = len(self.samples[0][1])
        else:
            self.fixed_caption_length = fixed_caption_length
    def __len__(self) -> int:
        return len(self.samples)
    def __getitem__(self, idx: int) -> tuple[torch.Tensor, torch.Tensor,
torch.Tensor]:
        img_filename, numerical_caption = self.samples[idx]
        image_path = os.path.join(self.images_dir_path, img_filename)
        try:
            image = Image.open(image_path).convert("RGB")
        except FileNotFoundError:
            print(f"Image file not found at {image_path}.")
        except Exception as e:
            print(f"Could not load image {image_path}.")
```

```
            raise
        if self.transform:
            image = self.transform(image)
        caption_tensor = torch.tensor(numerical_caption, dtype=torch.long)
        caption_input = caption_tensor[:-1]
        caption_target = caption_tensor[1:]
        return image, caption_input, caption_target
```

```
train_loader = DataLoader(
    dataset=train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=NUM_WORKERS,
    pin_memory=True
)

val_loader = DataLoader(
    dataset=val_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=NUM_WORKERS,
    pin_memory=True
)

test_loader = DataLoader(
    dataset=test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=NUM_WORKERS,
    pin_memory=True
)
```

دیتا ست های مورد نظر را مطابق تصویر بالا لود میکنیم. (Batch Size را برای همه آن ها ۳۲ در نظر می‌گیریم)

۱-۳: پیاده‌سازی مدل

در این بخش می خواهیم مدل ارائه شده در مقاله را مرحله به مرحله پیاده سازی کنیم. مدل ما از دو بخش اصلی Encoder و Decoder تشکیل می‌شود.

پیاده‌سازی بخش رمزگذار (Encoder):

برای این بخش از مدل از پیش آموزش داده شده ResNet50 استفاده می‌کنیم. لایه Fully Connected انتهایی مدل را که برای Classification اعمال شده را حذف کرده و بردار ویژگی های استخراج شده قبل از آن را به عنوان خروجی شبکه در نظر می گیریم (در پیاده سازی هنگام بارگذاری لایه ها، لایه آخر را در نظر نمی‌گیریم). این بردار ویژگی های استخراج شده از تصویر ورودی می باشد.

```python
class ResNet50Encoder(nn.Module):
    def __init__(self):

        super(ResNet50Encoder, self).__init__()
        resnet50 = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V1)
        modules = list(resnet50.children())[:-1]
        self.resnet_features = nn.Sequential(*modules)

        for param in self.resnet_features.parameters():
            param.requires_grad_(False)

    def forward(self, images: torch.Tensor) -> torch.Tensor:

        features = self.resnet_features(images)
        features = torch.flatten(features, start_dim=1)

        return features
```

شکل ۹: بخش **Encoder** (مدل **ResNet50**)

برای بررسی اندازه بردار خروجی، یک تصویر را به شبکه بالا اعمال میکنیم و سایز خروجی را مشاهده میکنیم:

```
Shape of input tensor: torch.Size([1, 3, 224, 224])
Shape of output feature vector: torch.Size([1, 2048])
```

سایز بردار خروجی ۲۰۴۸ می‌باشد.

پیاده‌سازی بخش رمزگشا:

در این بخش مطابق مقاله، از یک مدل ترکیبی LSTM-GRU برای رمزگشایی بردار ویژگی های ورودی و تولید کپشن استفاده می‌کنیم. این ساختار ترکیبی میتواند ویژگی های خوب هر دو مدل LSTM و GRU را برای ما حفظ کند و منجر به نتایج بهتری بشود.

در ابتدا از یک لایه Embedding برای تبدیل بردار های گسسته واژگان به بردار های پیوسته استفاده می کنیم. این لایه که در حین آموزش مدل پارامتر هایش تنظیم می شود، میتواند ارتباط واژگان با یکدیگر را تشخیص داده که باعث میشود علاوه بر کاهش بعد خروجی، روابط معنایی بین واژگان نیز حفظ بشود. سایز ورودی این لایه سایز دیکشنری و خروجی آن در مدل ما ۲۵۶ تعریف می شود.

سپس شبکه LSTM را تعریف میکنیم. سایز بردار های Cell state و Hidden state را ۵۱۲ در نظر می‌گیریم. مطابق مقاله، بردار ویژگی های تصویر را باید به عنوان Initial State به LSTM بدهیم. برا همین دو لایه Fully Connected استفاده میکنیم که بردار ورودی با سایز ۲۰۴۸ را گرفته و بردار خروجی با سایز ۵۱۲ تولید میکنند تا به عنوان حالت اولیه به LSTM داده شوند.

سپس شبکه GRU را با همان سایز ورودی ۵۱۲ استفاده می کنیم. مطابق مقاله، ورودی این شبکه، خروجی Cell State شبکه LSTM می‌باشد. خروجی تولید شده توسط این شبکه (GRU-hidden_state) خروجی دیکودر ما می باشد که سایز آن نیز ۵۱۲ تنظیم میشود.

از یک لایه Dropout در خروجی GRU قبل از اعمال آن به لایه خطی استفاده شده تا مدل دچار Overfitting بر روی یک سری ویژگی های خاص نشود. (نرخ Dropout برابر ۰.۵ میباشد)

در انتها از یک لایه خطی با سایز ورودی ۵۱۲ و سایز خروجی به اندازه دیکشنری استفاده شده است که با اعمال تابع SoftMax به خروجی آن میتوان کلمه بعدی با بیشترین احتمال را انتخاب کنیم.

```
class HybridLSTMGRUDecoder(nn.Module):
    def __init__(self, embed_size: int, hidden_size: int, vocab_size: int,
                 encoder_feature_dim: int = 2048, dropout_rate: float = 0.5):
        super(HybridLSTMGRUDecoder, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.init_h_lstm = nn.Linear(encoder_feature_dim, hidden_size)
        self.init_c_lstm = nn.Linear(encoder_feature_dim, hidden_size)
        self.lstm_cell = nn.LSTMCell(embed_size, hidden_size)
        self.gru_cell = nn.GRUCell(hidden_size, hidden_size)
        self.dropout = nn.Dropout(p=dropout_rate)
        self.fc_out = nn.Linear(hidden_size, vocab_size)
```

شکل ۱۰: ساختار دیکودر

ادغام Encoder و Decoder:

حال یک کلاس تعریف کرده و هر دو بخش رمزگذار و رمزگشا را ترکیب می‌کنیم. وزن های مدل CNN بخش رمزگذار را Freeze میکنیم تا حین آموزش تغییر نکنند.

نحوه عملکرد شبکه:

بردار ویژگی تصویر را به عنوان حالت اولیه به بخش LSTM اعمال میکنیم. سپس به تعداد کلمه های ورودی، هر بار یک کلمه را Embed کرده و به LSTM اعمال میکنیم. سپس خروجی LSTM به عنوان ورودی GRU اعمال شده و با عبور دادن خروجی GRU از لایه خطی انتهایی، کلمه بعدی را مدل حدس میزند. همین عمل را تکرار کرده تا تمام ۲۰ کلمه را مدل پیشبینی کند.

با توجه به ساختار توضیح داده شده، مدل به صورت end-to-end قابل آموزش می‌باشد (با اعمال یک Loss function و محاسبه گرادیان ها در مسیر Backward و آپدیت کردن وزن های شبکه)

```python
class HybridCaptioningModel(nn.Module):
    def __init__(self, embed_size: int, hidden_size: int, vocab_size: int,
                 encoder_feature_dim: int = 2048, dropout_rate: float = 0.5,
                 freeze_encoder: bool = True):
        super(HybridCaptioningModel, self).__init__()
        self.encoder = ResNet50Encoder()
        if not freeze_encoder:
            for param in self.encoder.parameters(): param.requires_grad_(True)
        else:
            for param in self.encoder.parameters(): param.requires_grad_(False)

        self.decoder = HybridLSTMGRUDecoder(
            embed_size=embed_size, hidden_size=hidden_size, vocab_size=vocab_size,
            encoder_feature_dim=encoder_feature_dim, dropout_rate=dropout_rate
        )
        self.vocab_size = vocab_size

    def forward(self, images: torch.Tensor, captions_in: torch.Tensor) -> torch.Tensor:
        image_features = self.encoder(images)
        return self.decoder(image_features, captions_in)
```

شکل ۱۱: ساختار شبکه نهایی

نکته: در هنگام لود کردن دیتا ست، دو بردار Caption_in و Caption_out را برای هر نمونه تشکیل میدهیم. از بردار اول برای آموزش استفاده میشود که شامل ۱۹ توکن ابتدایی کپشن است و با اعمال این توکن ها در هر مرحله، توکن بعدی را شبکه پیشبینی میکند و با تارگت مقایسه میکند.

۱-۴: آموزش و ارزیابی مدل

آموزش:

حال به آموزش مدل می پردازیم. تابع هزینه را CrossEntropy تعریف میکنیم (با توجه به اینکه جنس خروجی از نوع یک مسئله طبقه بندی است که توکن های دیکشنری کلاس های آن میباشند):

criterion = nn.CrossEntropyLoss(ignore_index=PAD_TOKEN_ID)

همچنین توکن های پدینگ را در محاسبه Loss نادیده می‌گیریم. این کار کمک میکند تا مدل پیشبینی Pad ها را یاد نگیرد. زیرا این توکن ها تنها در جهت برابر کردن سایز ورودی ها اضافه شده اند و در عمل مدل تا هنگام رسیدن به توکن <eos> باید کلمه بعدی را پیشبینی کند.

از بهینه سازی Adam با نرخ یادگیری 0.001 استفاده می‌کنیم:

optimizer = optim.Adam(caption_model.decoder.parameters(), lr=0.001)

اندازه Batch size را ۳۲ و تعداد Epoch های آموزش را ۴۰ تنظیم میکنیم (مطابق تنظیمات پیشنهاد شده در مقاله).

مدل روی Caption_in و تصویر ورودی Train شده و خروجی آن تولید میشود. Loss بر اساس خروجی تولید شده و Caption_out که خروجی مد نظر ما هست محاسبه شده سپس در مسیر Backward، وزن ها بروز می شوند. همچنین در با اعمال داده های ارزیابی به مدل در هر Epoch، Validation Loss را نیز محاسبه میکنیم.

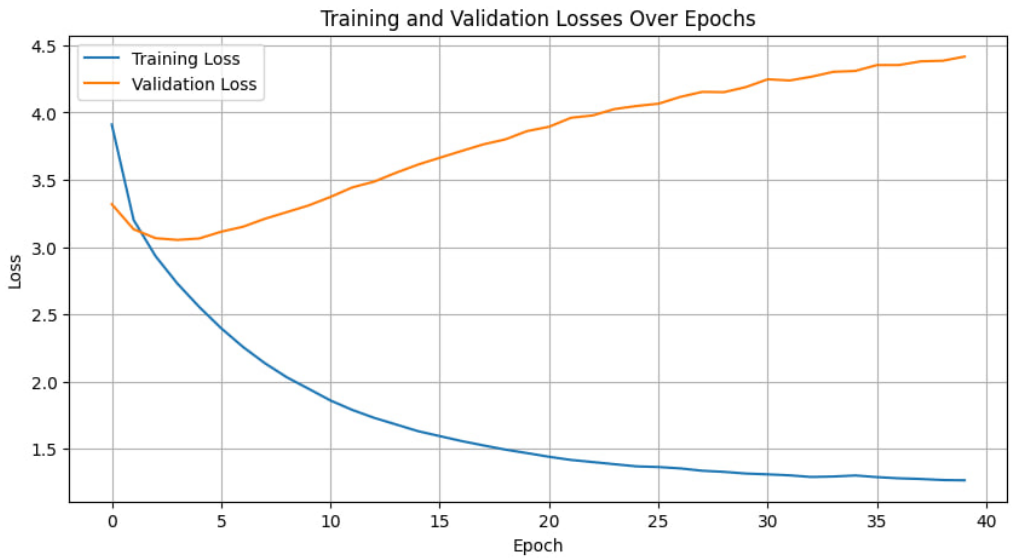
بهترین مدل را (بر اساس عملکرد آن بر روی داده های ارزیابی) ذخیره کرده تا از آن استفاده کنیم.

```
Starting training for 40 epochs on cuda...
  Epoch [1/40], Batch [100/1012], Train Loss: 4.5747
  Epoch [1/40], Batch [200/1012], Train Loss: 4.1514
  Epoch [1/40], Batch [300/1012], Train Loss: 4.1426
  Epoch [1/40], Batch [400/1012], Train Loss: 3.8342
  Epoch [1/40], Batch [500/1012], Train Loss: 3.8562
  Epoch [1/40], Batch [600/1012], Train Loss: 3.3232
  Epoch [1/40], Batch [700/1012], Train Loss: 3.8686
  Epoch [1/40], Batch [800/1012], Train Loss: 3.4306
  Epoch [1/40], Batch [900/1012], Train Loss: 3.3640
  Epoch [1/40], Batch [1000/1012], Train Loss: 3.2004
Epoch [1/40] completed in 3079.94s:
  Avg Training Loss: 3.9101, Avg Validation Loss: 3.3187
  Best model saved to /content/drive/My Drive/Colab Notebooks/HW4/best_hybrid_caption_model.pth (Val Loss: 3.3187)
  Epoch [2/40], Batch [100/1012], Train Loss: 3.1879
  Epoch [2/40], Batch [200/1012], Train Loss: 3.2018
  Epoch [2/40], Batch [300/1012], Train Loss: 3.2892
  Epoch [2/40], Batch [400/1012], Train Loss: 3.2124
  Epoch [2/40], Batch [500/1012], Train Loss: 3.0839
  Epoch [2/40], Batch [600/1012], Train Loss: 3.2927
  Epoch [2/40], Batch [700/1012], Train Loss: 3.0583
  Epoch [2/40], Batch [800/1012], Train Loss: 2.8846
  Epoch [2/40], Batch [900/1012], Train Loss: 3.4083
  Epoch [2/40], Batch [1000/1012], Train Loss: 3.1526
Epoch [2/40] completed in 296.56s:
```

شکل ۱۲: بخشی از **log** مدل حین آموزش

ارزیابی:

در انتها، نمودار خطای داده آموزش و ارزیابی را رسم میکنیم:



شکل ۱۳: نمودار خطای داده آموزش و ارزیابی

تحلیل نمودار:

همانطور که مبینیم در Epoch های اولیه (حدودا ۶)، هزینه داده های آموزش و ارزیابی همزمان کاهش می یابند که نشان دهنده یادگیری خوب مدل می باشد. ولی بعد از آن با وجود کاهش هزینه داده آموزش، هزینه داده های ارزیابی به شدت افزایش می یابد. این امر نشان دهنده بیش برازش (Overfitting) مدل بر روی داده های آموزش می باشد و گویی مدل دارد این داده ها را (همراه الگو ها و نویز هایشان) به حافظه میسپارد، به جای اینکه از روی داده ها یاد بگیرد.

یکی از دلایل بروز این اتفاق میتواند پیچیدگی بیش از حد نیاز مدلمان باشد. اگر چه مدل های بزرگ و پیچیده (مانند مدل ترکیبی ای که در اینجا پیاده سازی کردیم) به خاطر ساختار پیچیده و پارامتر های زیاد توانایی یادگرفتن الگو های پیچیده را دارند، ولی این امر زمانی اتفاق می افتد که داده متنوع و به اندازه کافی برای آموزششان داشته باشیم. حال آنکه در صورت استفاده از دیتا ست کوچک، این امکان وجود دارد که مدل به خاطر سایز بزرگش ورودی ها را به خاطر بسپارد (به جای یافتن الگو ها) و عملکرد آن بر روی داده هایی که تا به حال ندیده است کاهش یابد. دیتا ست ما شامل حدود ۶۰۰۰ تصویر به عنوان آموزش هست که هر عکس با ۵ کپشن مختلف به شبکه داده میشود. این امر اگرچه به نظر میرسد تعداد داده های ورودی را افزایش میدهد، ولی به هر حال ۵ تصویر تکراری هستند و نمیتوان انتظار داشت یادگیری خوبی را در مدل نتیجه دهد.

در بخش آخر مدل بهبود یافته ای را ارائه داده ایم که مشکلات بیش برازش را حل کنیم.

حال برای تولید خروجی ها مطابق مقاله، از دو الگوریتم Greedy search و Beam search استفاده می‌کنیم. در ادامه ابتدا این الگوریتم ها و نحوه پیاده سازی شان را توضیح میدهیم و سپس کپشن های تولید شده توسط هر کدام را بر روی تعدادی از تصاویر بررسی کرده و مقایسه می نماییم.

الگوریتم Greedy Search:

در این الگوریتم در هر لحظه برای پیشبینی کلمه بعد، تنها احتمالات خروجی حال حاضر بررسی میشود و کلمه با بیشترین احتمال انتخاب شده و سپس به سراغ پیشبینی کلمه بعدی می رود. در این الگوریتم توجهی به کلمات بعدی تولید شده و اینکه دنباله تولید شده در نهایت به چه صورت است توجهی نمی شود ولی از آن طرف سرعت بالایی در مرحله Inference دارد (هر بار یک توکن به شبکه اعمال شده و خروجی آن را به عنوان توکن بعدی ذخیره می کنیم)

نحوه پیاده سازی:

متد این الگوریتم در کلاس مدل نهایی پیاده سازی شده است. عملکرد آن به این صورت است که ابتدا تصویر را به مدل به عنوان حالت اولیه اعمال میکند. اولین توکن ورودی را <sos> به شبکه اعمال میکند و مرحله به مرحله توکن خروجی شبکه را ذخیره و آن را به عنوان ورودی در مرحله بعد اعمال میکند تا جایی که به توکن <eos> برسیم و کپشن کامل شود.

الگوریتم Beam Search:

این الگوریتم برای پیشبینی دنباله ای از کلمات که نسبت به دنباله های دیگر احتمال بیشتری دارد استفاده می‌گردد. پارامتر مهم در این الگوریتم Beam width می باشد که مشخص میکند در هر مرحله تولید دنباله در چه تعداد مسیر جدید (با چه عرضی) پیش‌روی کند. با هر بار پیشروی فرضیه های جدید تشکیل میشود که احتمال آن ها برابر با احتمال تا مرحله قبل به اضافه لگاریتم احتمال در مسیرهای جدید میباشد (دلیل اعمال لگاریتم، تبدیل محاسبات ضرب احتمالات در شاخه های مختلف به جمع میباشد.)

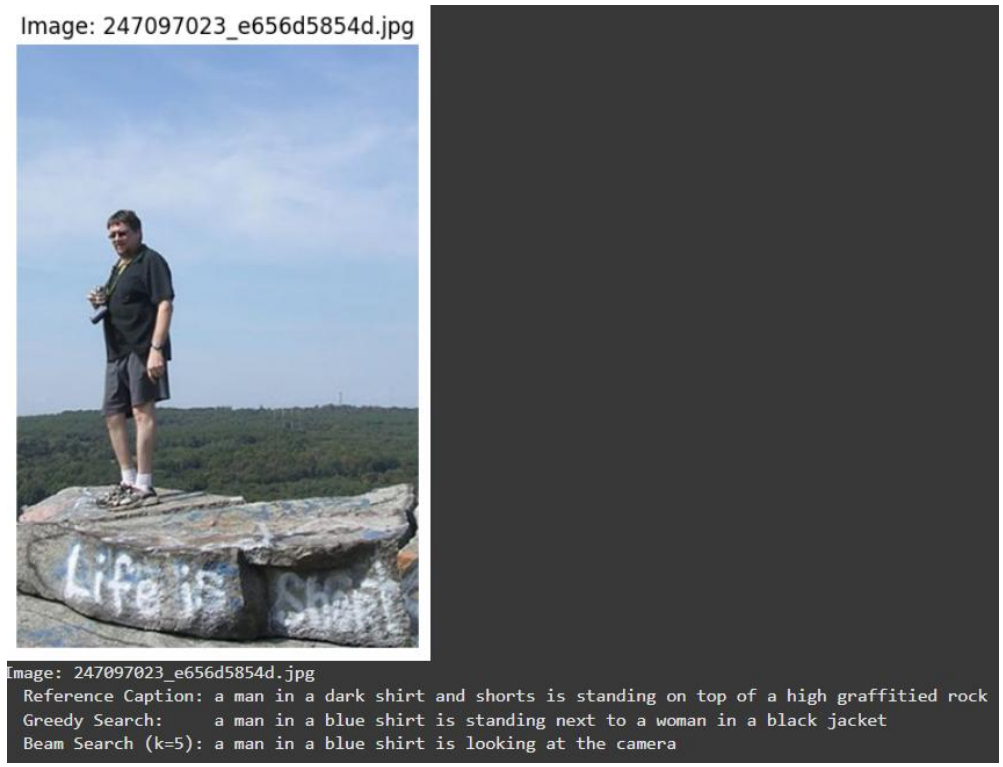
تا جایی ادامه میدهیم که به توکن <eos> در یک مسیر برسیم و یا طول آن به طول بیشینه برسد. سپس همه فرضیه های ایجاد شده را بررسی کرده و فرضیه با بیشترین احتمال را انتخاب میکنیم.

مقایسه: اگرچه الگوریتم Beam search نسبت به Greedy حجم محاسبات بالاتری دارد،به دلیل بررسی مسیرهای مختلف، شانس بیشتری دارد که دنباله های بهتر را پیدا کند که از نظر معنایی مرتبط اند.

حال هر دو الگوریتم را به تعدادی از تصاویر دادگان تست اعمال کرده و نتایج را بررسی می کنیم (برای الگوریتم Beam، Width را برابر ۵ قرار میدهیم زیرا در مقاله اشاره شده بود که طبق بررسی هایی که انجام داده اند و در نظر گرفتن tradeoff ها، عرض بهینه ای می باشد):

شکل ۱۴: خروجی اول شبکه

شکل ۱۵: خروجی دوم شبکه



شکل ۱۶: خروجی سوم شبکه

شکل ۱۷: خروجی چهارم شبکه

شکل ۱۸: خروجی پنجم شبکه

شکل ۱۹: خروجی ششم شبکه

شکل ۲۰: خروجی هفتم شبکه

از نتایج بالا مشخص است که الگوریتم Beam به مراتب خروجی های بهتری نسبت به Greedy تولید می‌کند.

شناسایی خطاها:

برخی از خطاها در شکل های بالا مشهود هستند.

برای مثال در تصویر خروجی سوم رنگ لباس مرد اشتباها آبی تشخیص داده شده که میتواند به خاطر رنگ غالب آسمان در تصویر آبی هست باشد.

در خروجی چهارم شبکه، به دلیل تعداد زیاد افراد در تصویر، کپشن آن میگوید که "تعداد زیادی آدم در زمین ایستاده اند" ولی ویژگی اصلی تصویر که افراد را در حال بازی نشان میدهد تشخیص نداده است.

در خروجی ششم، تعداد سگ ها را نتوانسته تشخیص بدهد و به اشتباه گفته که سگ ها در زمین چمن هستند (در حالی که در آب بازی می کنند). این میتواند به دلیل حافظه مدل از تصاویر دیگر که سگی در چمن بازی میکند ایجاد شده باشد و به نوعی Overfitting بر روی این نوع ورودی ها رخ داده باشد.

شکل ۲۱: خروجی هشتم شبکه

در خروجی هشتم، به اشتباه امده است که یک مرد در مقابل جمعیتی ایستاده است. این امر میتواند به خاطر تشخیص اشتباه درختان به عنوان انسان رخ داده باشد.

۵-۱: امتیازی

برای ارزیابی مدل‌های تولید کپشن از معیار هایی مانند BLEU، METEOR، ROUGE، SPICE و ... استفاده می شود.

معیار BLEU(Bilingual Evaluation Understudy) شباهت کپشن های تولید شده را با کپشن های مرجع محاسبه می کند. N-gram نشان دهنده دنباله متوالی از کلمات است که به عنوان یک بخش در نظر گرفته میشود. برای مثال جمله "هوا خوب است" با n-gram =1 به صورت "هوا " ، "خوب" ، " است" و با n-gram=2 به صورت "هوا خوب" ، " خوب است" در می آید.

برای هر نوع n-gram، معیار BLEU مشخص میکند که چه تعداد از n-gram ها در حداقل یکی از کپشن های مرجع تصویر یافت میشوند و یک امتیاز بین ۰ تا ۱۰۰ میدهد. امتیاز ۱۰۰ نشان دهنده تطابق کامل کپشن تولید شده با کپشن های مرجع می باشد.

معیار های BLEU-1 (n-gram=1) تا BLEU-4 (n-gram=4) را محاسبه میکنیم. معیار BLEU-1 بیشتر وجود کلمات کلیدی و مطابق با کپشن های مرجع را می‌سنجد و به ارتباط بین کلمات جمله توجهی نمی کند. ولی معیار BLEU-4 از آن طرف، به دنبال ساختار های پیچیده تر میگردد و میتواند معیار بهتری برای بررسی روان بودن و انسجام کپشن ها را ارائه بدهد.

جدول زیر متریک های بدست آمده بر روی مدل مان را برای دو الگوریتم Greedy و Beam برروی مجموعه داده تست نشان میدهد:

جدول ۱: معیارهای **BLEU** برروی مجموعه داده تست

	Greedy Seach	**Beam Search (Width = 5)**
BLEU-1	45.41	48.10
BLEU-2	28.02	29.54
BLEU-3	17.47	18.75
BLEU-4	11.18	12.17

مشاهده میکنیم که معیار BLEU-1 با مقدار گزارش شده در مقاله (۶۰ درصد) فاصله دارد. در بخش بعد با بهبود معماری و پارامتر های یادگیری مدل، سعی می کنیم نتایج را بهبود دهیم.

همچنین عملکرد بهتر الگوریتم Beam Search نسبت به Greedy مشخص می شود. (با این که معیار های هر دو نزدیک بهم هستند ولی Beam ساختار های منسجم تری را نمونه های نشان داده شده در بخش قبل از خود نشان داد.

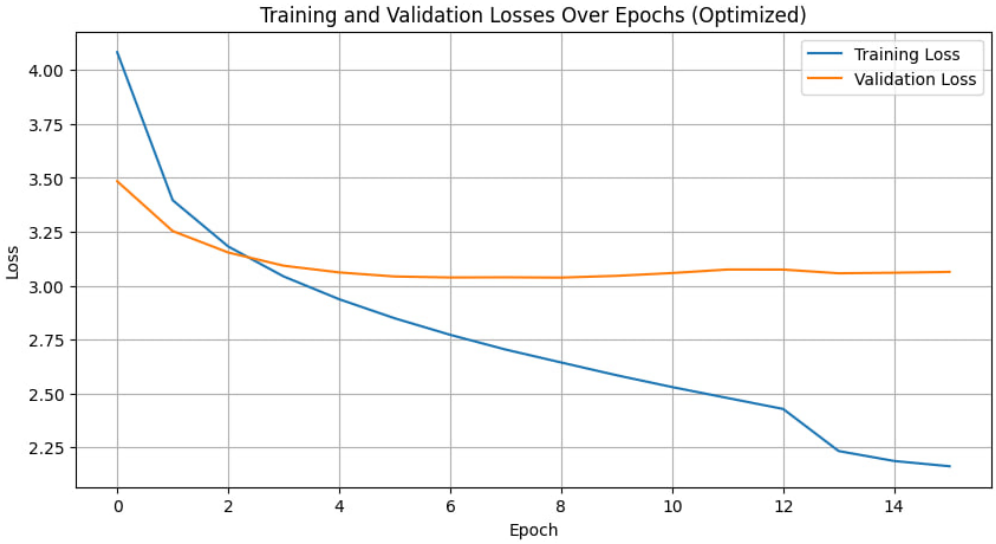
بهبود مدل:

در بخش قبل دیدیم که مدل ما دچار Overfit شد و نمودار خطای داده های ارزیابی این موضوع را به وضوح نشان میداد. حال برای بهبود مدل تغییراتی در معماری و پارامترهای آموزش میدهیم.

اول از همه سایز لایه Embedding و سایز Hidden State ها را از ۵۱۲ به ۲۵۶ کاهش میدهیم. این امر باعث کوچک تر شدن مدل شده و احتمال Overfit شدن را کاهش میدهد.

تنظیمات اپتیمایزر را تغییر داده و Weight decay با مقدار 10^-5 را اعمال میکنیم.

همچنین نرخ یادگیری را متغیر قرار داده که با نسبت یک دهم در صورتی که خطا کاهش نیابد، کوچک میکند.



شکل ۲۲: نمودار خطای داده آموزش و ارزیابی مدل بهینه شده

همانطور که میبینیم مقدار خطای شبکه در این وضعیت نیز از ۳ پایینتر نیامد (مشابه حالت قبل) ولی مدل دچار بیش برازش نشد و مقدار خطای داده های ارزیابی حول همان ۳ باقی ماند.

برای مقایسه بهتر این مدل با حالت قبل، متریک های BLEU را برروی داده های تست بدست می آوریم.

جدول ۲: معیارهای **BLEU** برروی مجموعه داده تست برای مدل بهبود یافته

	Greedy Seach	**Beam Search (Width = 5)**
BLEU-1	53.02	55.2
BLEU-2	34.93	36.44
BLEU-3	22.42	23.58
BLEU-4	14.52	15.33

همانطور که میبینیم در مقایسه با مدل قبلی، برای الگوریتم Beam، شاخص BLEU-1 حدود ۷ درصد بهبود یافته است و به نتیجه مقاله (۶۰ درصد) تا حد خوبی نزدیک شده است.

تعدادی از کپشن های تولید شده توسط این مدل را در تصاویر زیر میتوان مشاهده نمود.

شکل ۲۳: خروجی اول مدل بهبود یافته

شکل ۲۴: خروجی دوم مدل بهبود یافته

شکل ۲۵: خروجی سوم مدل بهبود یافته

شکل ۲۶: خروجی چهارم مدل بهبود یافته

شکل ۲۷: خروجی پنجم مدل بهبود یافته

تحلیل: مشاهده می شود که اگر چه مدل هنوز در تشخیص برخی اشیا مشکل دارد (مانند خروجی پنجم) ولی کپشن های منسجم تری به طور کلی تولید شده است. به خصوص خروجی هایی که با الگوریتم Greedy تولید شده اند در مقایسه با حالت قبل بهبود قابل ملاحظه ای یافته اند. (در حالت قبل خروجی های الگوریتم Greedy عمدتا توصیف خوبی از عکس نمی‌کردند)

پرسش ۲ – پیش بینی سری زمانی برای Clinical Event

بخش اول: متدولوژی

مدل های مختلف در پیشبینی زمانی

د پیشبینی زمانی در واقع مسئله ای است که هدف آن تخمین رخداد ها یا مقادیر در آینده بر اساس رخداد ها و یا مقادیر گذشته است. البته در برخی موارد ما می توانیم از اطلاعات آینده نیز برای تخمین استفاده کنیم! برای مثال ممکن است در یک گزارش پزشکی ذکر شود که بیمار دچار فشار خون بالا بوده و پس از یک هفته سکته کرده است و از ما خواسته شود که رخداد هایی که در این هفته اتفاق افتاده است را پیشبینی کنیم. در این صورت ما می توانیم از اطلاعات گذشته و آینده استفاده کنیم! برای انجام پیشبینی زمانی مدل های مختلفی وجود دارند. در بخش Related Work مقاله ضمیمه شده در دستور کار، موارد زیر ذکر شده‌اند(با توجه به اینکه موارد مربوط به Methodology را در بخش های بعدی بررسی میکنیم، در این بخش مدل های مربوط به Related Work را بررسی کردیم):

۱- **مدل مارکوف (Markov State Model)** : این مدل صرفاً از وضعیت اخیر (آخرین بردار رخدادها) برای پیش‌بینی استفاده می‌کند و فرض می‌کند که تنها وضعیت فعلی برای تعیین آینده کافی است.

$$p(y_1, y_2, \dots, y_T) = p(y_1) \prod_{t=2}^{T} p(y_t | y_{t-1})$$

۲- **مدل مارکوف پنهان (Hidden Markov Model (HMM))** : در این نوع مدل ها چیزی تعریف می شود به نام وضعیت پنهان(Hidden state). وضعیت پنهان در واقع خلاصه ای از دنباله تمام رخداد های قبلی است و روی تصمیم گیری مدل اثر گذار است.

$$A_{i,j} = P(\, z_t = j \mid z_{t-1} = i \,)$$

$$B_{i,j} = P(\, y_t = j \mid z_t = i \,)$$

Aماتریس انتقال بین وضعیت‌های پنهان و B ماتریس انتشار (emission) برای تولید مشاهدات از حالت‌های پنهان است.

۳- **سیستم دینامیکی خطی (Linear Dynamical System (LDS))** : یکی از چالش های مدل های HMM این بود که وضعیت پنهان متغییری گسسته است. یعنی وضعیت پنهان بیمار برای مثال «بد»، «خوب» یا «وخیم» است. این مورد باعث محدودیت مدل HMM میشد که در مدل های LDS وضعیت پنهان به شکل پیوسته تعریف شد و این مشکل برطرف شد.

۴- **مدل‌های پیوسته‌زمان (Continuous Time Models)** : اغلب مدل های پیشبینی زمانی زمان-گسسته هستند و زمان را به گام های مشخص تقسیم میکنند در صورتی که در دنیای وقاعی ممکن است ثبت رخداد ها در گام های مشخص اتفاق نیوفتد و نامنظم باشد. به منظور حل این چالش مدل هایی مثل فرایند گوسی(Gaussian Process) طراحی شدند که پیشبینی زمانی را به صورت زمان- پیوسته انجام دهند.

۵- **مدل‌های مبتنی بر شبکه عصبی(Neural-based Models)** : مدل‌های مبتنی بر شبکه‌های عصبی بازگشتی (RNN) در مسائل سری‌زمانی مانند HMM ها از طریق نگهداری و به‌روزرسانی یک بردار «وضعیت پنهان» در هر گام زمانی عمل می‌کنند. این بردار در تصمیم‌گیری گام بعدی مورد استفاده قرار می‌گیرد. مدل‌های RNN کلاسیک در یادگیری وابستگی‌های بلندمدت دچار مشکل نابودی گرادیان (vanishing gradient) هستند. این مشکل یعنی در حین یادگیری گرادیان به گام های زمانی اول نمیرسد و یادگیری به درستی انجام نمیشود(انگار رخداد های اولیه تاثیر خیلی کمی روی رخداد آینده دارند).

۶- **LSTM (Long Short-Term Memory)** : برای حل مشکل نابودی گرادیان این مدل طراحی شد. این مدل با استفاده از دروازه‌های کنترل حافظه، قادر به حفظ وابستگی‌های بلندمدت است. در این پیاده سازی ما از این مورد استفاده خواهیم کرد.

روش State-Space Markov Event prediction

در این روش در هر لحظه t وضعیت رخداد ها (مثلا تجویز دارو، آزمایش ها، علائم و...) به صورت یک بردار باینری که هر درایه می تواند صفر یا یک باشد و نشانگر رخ دادن یا ندادن یک رخداد باشد، نشان داده می شود(البته در برخی متغییر ها این مقدار ۰ یا ۱ نیست مثلا فشار خون). در State-Space Markov Event prediction هدف ما این است که با استفاده از وضعیت در لحظه t وضعیت در لحظه t+1 را پیشبینی کنیم. برای اینکار مدل با استفاده از ماتریس وزن ها و بردار بایاس، به شکل زیر یاد میگیرد که چطور وضغیت فعلی به وضعیت بعدی منجر می‌شود:

$$\widehat{y_{t+1}} = \sigma(W \cdot y_t + b)$$

استفاده از تابع سیگموید به این دلیل است که خروجی بین ۰ و ۱ باشد و بتوان آن را به عنوان احتمال رخداد تفسیر کرد. با توجه به این موضوع از تابع سیگموید به عنوان تابع فعالسازی این مدل استفاده می‌شود.(مدل یک بار با این تابع فعالسازی و یک بار با Relu تست شد، با توجه به اینکه نتایج Relu بهتر بود از Relu استفاده کردیم.) از تابع هزینه Binary Cross Entropy برای این مدل استفاد ه می شود.

$$L = \sum_t -[y_t \cdot \log(\hat{y_t}) - (1 - y_t) \cdot \log(1 - \hat{y_t})]$$

در مقاله learing rate = 0.005 ذکر شده و از بهینه سازی مبتنی بر sgd استفاده شده است. همچنین از dropout نیز برای جلوگیری از overfitting با p = 0.5 استفاده شده است. در مقاله صحبتی از مجاز بودن یا نبودن لایه های نرمالیزیشن نشده است. ولی اگر ورودی ها به صورت بردار های چند متغییره دودویی باشند، نرمال کردن داده ها آنها را خراب میکند ولی اگر داده ها پیوسته ای باشند می توانیم آنها را نرمالایز کنیم . البته با توجه به اینکه مدل ها چندان عمیق نیستند و در ابتدا داده ها را نرمالایز میکنیم چندان نیازی هم به این لایه ها نیست. البته در این بخش صرفا این موارد را ذکر میکنیم و در بخش آموزش مدل آنها را گزارش میکنیم.

Summary مدل پیاده سازی شده به این شکل است(با توجه به بخش تحلیل آماری ۲۰ ویژگی داریم + ساعت که مجموعا شامل ۲۱ ویژگی میشود):

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MarkovMLP                                [64, 21]                  --
├─Sequential: 1-1                        [64, 21]                  --
│    └─Linear: 2-1                       [64, 128]                 2,816
│    └─ReLU: 2-2                         [64, 128]                 --
│    └─Linear: 2-3                       [64, 21]                  2,709
==========================================================================================
Total params: 5,525
Trainable params: 5,525
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.35
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.08
Params size (MB): 0.02
Estimated Total Size (MB): 0.10
==========================================================================================
```

دقت کنیم با توجه به اینکه خروجی های ما مقادیر پیوسته هستند، در لایه آخر MLP از تابع فعال سازی استفاده نمی کنیم تا مدل بتواند مقادیر پیوسته را پیشبینی کند. در مدل های بعدی نیز این کار را انجام میدهیم. همچنین باز بخاطر عمیق نبودن مدل از dropout استفاده نمیکنیم.

روش LSTM-based event prediction

در این مدل به عنوان ورودی یک بردار مانند m_t داریم که تعدادی از درایه های آن یک است که نشان دهنده رخ دادن یک اتفاق است. در روش LSTM ابتدا این بردار را با یک تبدیل خطی به یک بردار چگال تر به نام x_t تبدیل میکنیم:

$$x_t = W_{\text{emb}} \cdot m_t$$

از این به بعد منظور ما از ورودی همان x_t است.

در مدل LSTM همانطور که در کلاس و اسلاید ها مطرح شده ما یک hidden state و یک cell state داریم. نوشتن و فراموش کردن یک داده روی cell state توسط برخی گیت ها کنترل میشود. وضعیت

پنهان در این مدل از این cell satae استفاده می کند و همچنین ورودی این لحظه و وضعیت پنهان لحظه قبل از طریق یک gate به نام output gate روی وضعیت پنهان لحظه کنونی تاثیر میگذارند. نهایتا وضعیت پنهان و ورودی این لحظه خروجی را می سازند. فرمول های مربوط به این مدل در زیر آورده شده:

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$\widetilde{C_t} = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C_t}$$
$$h_t = o_t \odot \tanh(C_t)$$
$$\widehat{y_{t+1}} = \sigma(W_{\text{fc}} \cdot h_t + b_{\text{fc}})$$

f_t, i_t, o_t به‌ترتیب دروازه‌های خروجی، ورودی، فراموشی هستند.

یک Forward pass از LSTM شامل این مراحل است:

۱- Embedding ورودی ها:

بردار ورودی multi-hot مربوط به رخدادهای لحظه‌ی t، با استفاده از یک ماتریس Embedding خطی به یک بردار عددی چگال تبدیل می‌شود:

$$x_t = W_{\text{emb}} \cdot m_t$$

۲- محاسبه gate ها:

سه gate اصلی LSTM با استفاده از بردار ورودی x_t و حالت پنهان قبلی h_{t-1} به‌صورت زیر محاسبه می‌شوند:

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

دقت کنید که این محاسبات می توانند موازی با هم انجام شوند.

۳- تولید Candidate Cell State:

حافظه‌ی جدید پیشنهادی برای ورود به سلول محاسبه می‌شود.

$$\widetilde{C_t} = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

۴- به‌روزرسانی حافظه سلولی:

با ترکیب اطلاعات حافظه‌ی قبلی و حافظه‌ی جدید پیشنهادی، مقدار جدید حافظه سلولی محاسبه می‌شود:

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C_t}$$

۵- حاسبه حالت پنهان جدید(Hidden State) :

خروجی LSTM در زمان t از حافظه‌ی سلولی فعلی و دروازه خروجی به‌دست می‌آید:

$$h_t = o_t \odot \tanh(C_t)$$

۶- پیش‌بینی رخدادهای آینده:

در نهایت با استفاده از h_t، احتمال وقوع هر رخداد در گام زمانی بعدی از طریق یک لایه Fully Connectedبا تابع فعال‌سازی سیگموید محاسبه می‌شود:

$$\widehat{y_{t+1}} = \sigma(W_{\text{fc}} \cdot h_t + b_{\text{fc}})$$

GRU, Bi-Directional GRU, LSTM & Bi-Directional LSTM

یک لایه LSTM استاندارد، دنباله ورودی را تنها در یک جهت، (معمولاً از ابتدا به انتها) پردازش می‌کند. این بدان معناست که در هر گام زمانی t، حالت پنهان (hidden state) محاسبه‌شده تنها حاوی اطلاعاتی از ورودی‌های گذشته (گام‌های زمانی 1تا t-1)و ورودی فعلی (گام زمانی t)است.

در مقابل، یک لایهBi-Directional LSTM از دو لایه LSTM مجزا تشکیل شده است:

۱. **لایه LSTM روبه‌جلو :(Forward LSTM)** این لایه دنباله ورودی را به ترتیب زمانی استاندارد، از x_1تا x_T، پردازش می‌کند. در هر گام زمانی t، این لایه یک بردار حالت پنهان روبه‌جلو تولید می‌کند که خلاصه‌ای از اطلاعات گذشته و حال است.

۲. **لایه LSTM روبه‌عقب (Backward LSTM) :**این لایه دنباله ورودی را به ترتیب زمانی معکوس، از x_T تا x_1، پردازش می‌کند. در هر گام زمانی t، این لایه یک بردار حالت پنهان روبه‌عقب تولید می‌کند که خلاصه‌ای از اطلاعات آینده (نسبت به جهت پردازش استاندارد) و حال است.

خروجی نهایی لایه BiLSTM در هر گام زمانی t، از ترکیب این دو بردار حالت پنهان به دست می‌آید. رایج‌ترین روش ترکیب، concatenation دو بردار است. ولی روش‌های دیگری مانند جمع، میانگین‌گیری یا ضرب نیز می‌توانند مورد استفاده قرار گیرند.

مزیت اصلی و بنیادین استفاده از BiLSTM در مقایسه با LSTM استاندارد، دسترسی به اطلاعات زمینه‌ای کامل‌تر در هر گام زمانی است.

۱. **درک عمیق‌تر از دنباله:** با پردازش دنباله در هر دو جهت، مدل قادر است وابستگی‌ها و الگوهایی را که ممکن است در یک پردازش تک‌جهته نادیده گرفته شوند، شناسایی کند. به عنوان مثال، در تحلیل داده‌های بیمار، یک رویداد یا مقدار خاص ممکن است نه تنها تحت تأثیر رویدادهای قبلی، بلکه پیش‌بینی‌کننده رویدادهای آتی نیز باشد (یا اهمیت آن با توجه به رویدادهای آتی مشخص شود).

۲. **بهبود عملکرد در وظایف خاص:** در بسیاری از کاربردها، مانند تحلیل احساسات متن، برچسب‌گذاری اجزای کلام، یا حتی در برخی جنبه‌های تحلیل سری‌های زمانی (مانند imputation یا طبقه‌بندی کل دنباله)، دسترسی به زمینه کامل منجر به عملکرد بهتر مدل می‌شود. زیرا تصمیم‌گیری در مورد یک عنصر خاص در دنباله با آگاهی از عناصر قبل و بعد از آن، دقیق‌تر خواهد بود.

لازم به ذکر است که در وظایف پیش‌بینی سری‌های زمانی که هدف، پیش‌بینی مقادیر آینده بر اساس مقادیر گذشته است، استفاده مستقیم از اطلاعات "آینده واقعی" (که هنوز رخ نداده) امکان‌پذیر نیست. با این حال، BiLSTM می‌تواند در بخش‌هایی از مدل‌های پیچیده‌تر (مانند معماری‌های encoder-decoder برای پیش‌بینی دنباله به دنباله) یا برای استخراج ویژگی‌های بهتر از یک پنجره زمانی مشخص که کل آن در دسترس است، مفید واقع شود.

Summary مدل های پیاده سازی شده در زیر آمده است.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
LSTMModel                                [64, 21]                  --
├─LSTM: 1-1                              [64, 6, 128]              77,312
├─Sequential: 1-2                        [64, 21]                  --
│    └─Linear: 2-1                       [64, 128]                 16,512
│    └─ReLU: 2-2                         [64, 128]                 --
│    └─Linear: 2-3                       [64, 21]                  2,709
==========================================================================================
Total params: 96,533
Trainable params: 96,533
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 30.92
==========================================================================================
Input size (MB): 0.03
Forward/backward pass size (MB): 0.47
Params size (MB): 0.39
Estimated Total Size (MB): 0.89
==========================================================================================
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
BiLSTMModel                              [64, 21]                  --
├─LSTM: 1-1                              [64, 6, 256]              154,624
├─Sequential: 1-2                        [64, 21]                  --
│    └─Linear: 2-1                       [64, 128]                 32,896
│    └─ReLU: 2-2                         [64, 128]                 --
│    └─Linear: 2-3                       [64, 21]                  2,709
==========================================================================================
Total params: 190,229
Trainable params: 190,229
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 61.65
==========================================================================================
Input size (MB): 0.03
Forward/backward pass size (MB): 0.86
Params size (MB): 0.76
Estimated Total Size (MB): 1.66
==========================================================================================
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
GRUModel                                 [64, 21]                  --
├─GRU: 1-1                               [64, 6, 128]              57,984
├─Sequential: 1-2                        [64, 21]                  --
│    └─Linear: 2-1                       [64, 128]                 16,512
│    └─ReLU: 2-2                         [64, 128]                 --
│    └─Linear: 2-3                       [64, 21]                  2,709
==========================================================================================
Total params: 77,205
Trainable params: 77,205
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 23.50
==========================================================================================
Input size (MB): 0.03
Forward/backward pass size (MB): 0.47
Params size (MB): 0.31
Estimated Total Size (MB): 0.81
==========================================================================================
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
BiGRUModel                               [64, 21]                  --
├─GRU: 1-1                               [64, 6, 256]              115,968
├─Sequential: 1-2                        [64, 21]                  --
│    └─Linear: 2-1                       [64, 128]                 32,896
│    └─ReLU: 2-2                         [64, 128]                 --
│    └─Linear: 2-3                       [64, 21]                  2,709
==========================================================================================
Total params: 151,573
Trainable params: 151,573
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 46.81
==========================================================================================
Input size (MB): 0.03
Forward/backward pass size (MB): 0.86
Params size (MB): 0.61
Estimated Total Size (MB): 1.50
==========================================================================================
```

بخش دوم: آماده سازی داده ها و تحلیل آماری

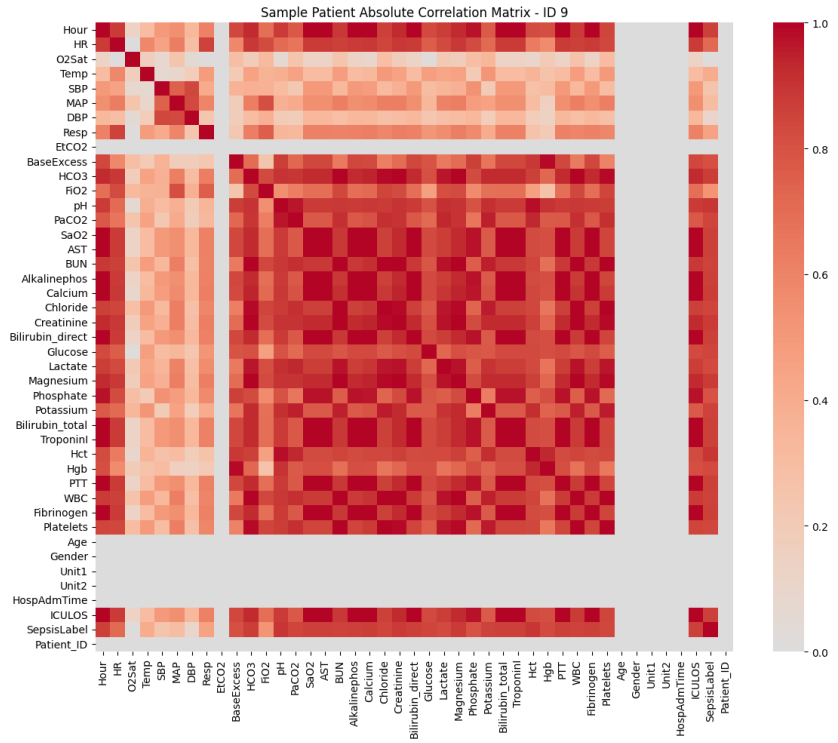
Correlation Matrix

پنج سطر اول داده های ترین را نشان میدهیم:

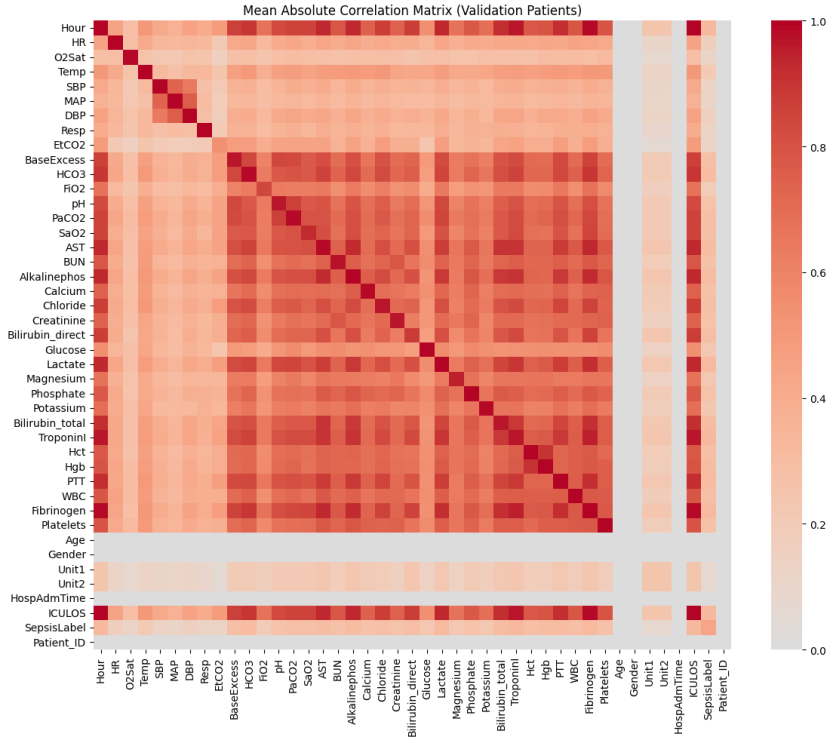
جدول ۳پنج نمونه اول داده های ولیدیشن

	Unnamed: 0	Hour	HR	O2Sat	Temp	SBP	MAP	DBP	Resp	EtCO2	...	Fibrinogen	Platelets	Age	Gender	Unit1	Unit2	HospAdmTime	ICULOS	SepsisLabel	Patient_ID
0	236	236	128.0	96.0	36.2800	149.0	113.0	92.0	28.5	33.029605	...	378.146104	726.000000	27.92	1	0.0	1.0	-0.03	237	0	9
1	237	237	129.0	94.0	37.0425	136.0	102.0	83.0	28.0	33.029605	...	376.746753	729.666667	27.92	1	0.0	1.0	-0.03	238	0	9
2	238	238	133.0	94.0	37.8050	141.0	106.0	86.0	33.0	33.029605	...	375.347403	733.333333	27.92	1	0.0	1.0	-0.03	239	0	9
3	239	239	137.0	94.0	38.5675	142.0	106.0	87.0	30.0	33.029605	...	373.948052	737.000000	27.92	1	0.0	1.0	-0.03	240	0	9
4	240	240	138.0	96.0	39.3300	142.0	108.0	87.0	30.5	33.029605	...	372.548701	740.666667	27.92	1	0.0	1.0	-0.03	241	0	9

برای بدست آوردن Correlation Matrix ابتدا سری زمانی مربوط به هر بیمار را جدا میکنیم. در هر سری زمانی Correlation Matrix را محاسبه میکنیم و نهایتا از این ماتریس ها میانگین میگیریم. دقت کنیم که پس از محاسبه ماتریس کرولیشن برای هر بیمار از ماتریس کرولیشن قدرمطلق میگیریم. زیرا کرولیشن منفی نیز دقیقا به اندازه کرولیشن مثبت نشان دهنده اشتراک اطلاعات است و اگه قدر مطلق نگیریم ماتریس ماینگین را دچار مشکل خواهد کرد. در زیر ماتریس همبستگی میانگین و ماتریس همبستگی مربوط به یک بیمار آورده شده است.



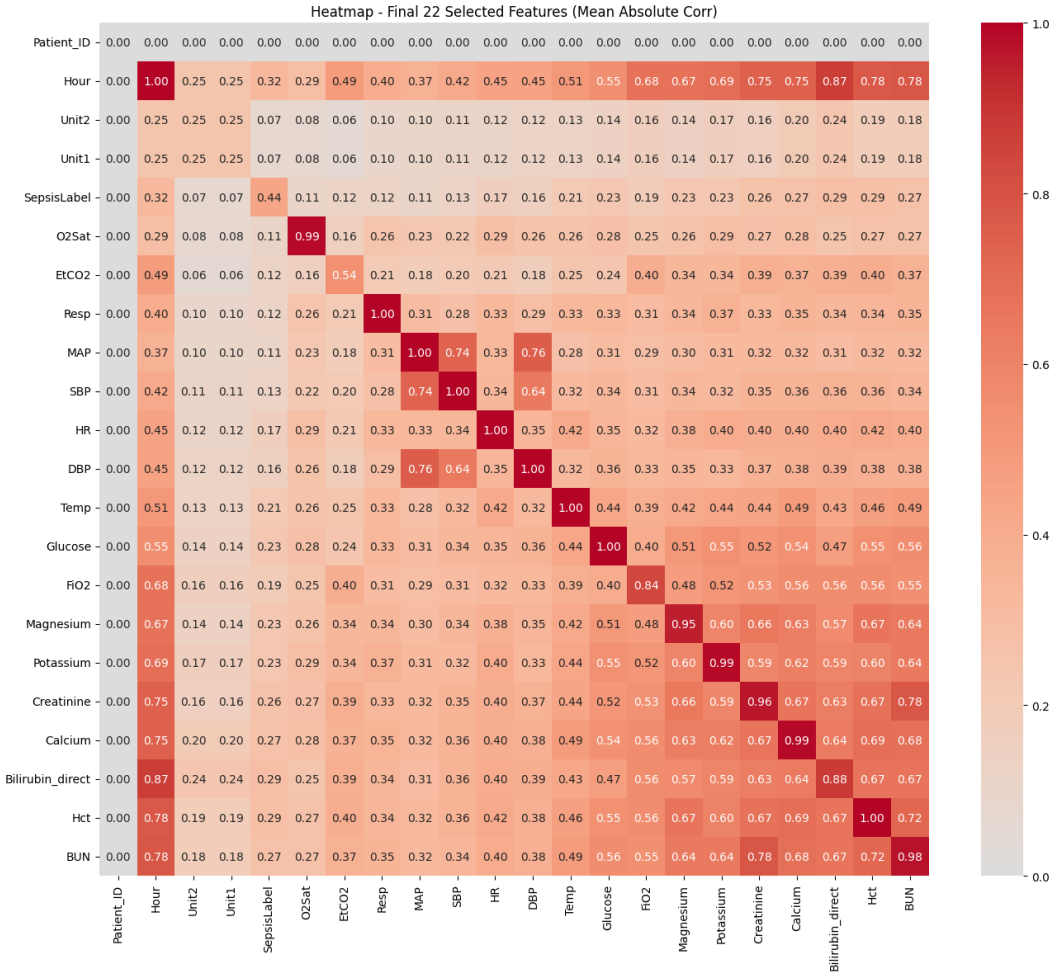
شکل ۲۸ ماتریس همبستگی مربوط به بیمار۹



شکل **29** ماتریس همبستگی میانگین

با توجه به ماتریس های بالا، مقادیر Age و Gender برای یک بیمار همواره ثابت است(مثل patient_ID). بنابرین عملا نیاز به پیشبینی خاصی ندارد و می توان این موارد را حذف کرد. hospAdmTime نیز زمان شروع بستری بیمار است که باز ثابت و قابل حذف است.موارد unit1 و unit2 نیز بخشی است که بیمار در آن بستری شده و در طول فرایند بستری بودن بیمار ثابت است. ممکن است با توجه به اینکه ثابت است بنظر برسد که قابل حذف است ولی با توجه به اینکه می تواند برای ویژگی های دیگر مفید باشد، این مقادیر را فعلا نگه میداریم.

در مرحله بعدی میگردیم و ویژگی هایی که بیشترین کرولیشن(کرولیشن بالای ۰.۹) را باهم دارند پیدا میکنیم و از بین ویژگی هایی که بیشترین کرولیشن را باهم دارند، ویژگی که کمترین کرولیشن با بقیه ویژگی ها دارد را نگه میداریم و دیگری را حذف میکنیم. نهایتا پس از حذف این ویژگی ها از بین ویژگی های باقی مانده ۲۰ ویژگی که کمترین کرولیشن با یکدیگر دارند را انتخاب میکنیم.(در واقع به علت اینکه hour و patient_ID را نگه میداریم یک ماتریس ۲۲ در ۲۲ داریم).



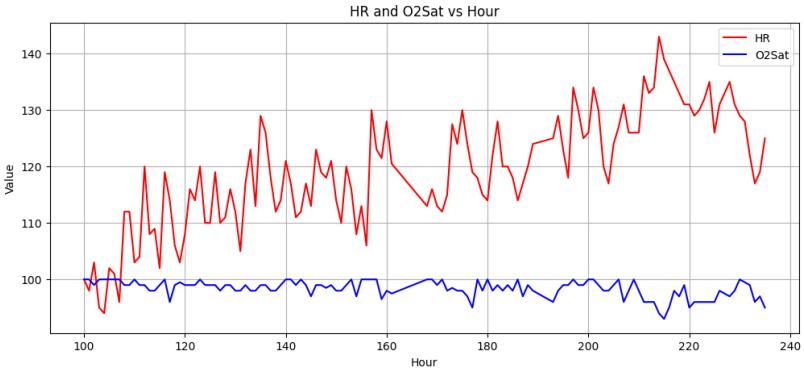
شکل ۳۰ ویژگی های باقی مانده نهایی

علت اینکه تاکیید روی کمترین کرولیشن است این است که می خواهیم ستون هایی را نگه داریم که واقعا دارای اطلاعات جدید باشند. بنابرین ستون هایی که کرولیشن زیادی باهم دارند را حذف میکنیم زیرا انگار با داشتن یکی از آن ستون ها، اطلاعات ستون دیگر را نیز داریم.

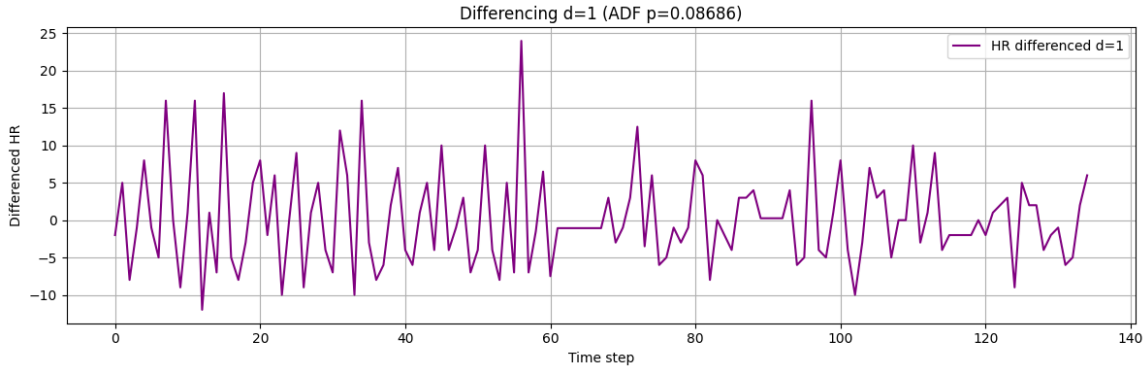
نهایتا داده های هر مریض را جدا میکنیم و در یک فایل csv جداگانه ذخیره میکنیم. از هر مریض نیز صرفا ۲۰ ویژگی با کمترین کرولیشن و ساعت را نگه میداریم.

SARIMAX

برای این بخش ابتدا میگردیم و بیماری که بیشترین داده ها را در دسته داده های ترین دارد پیدا میکنیم. سپس ستون HR و O2Sat (ورودی بیرونی)را بر حسب ستون ساعت برای این بیمار، در یک نمودار رسم میکنیم. تست ADF را روی نمودار HR انجام میدهیم. در سری اول مقدار p از ۵٪ کمتر نمی شود و ما مجبور به مشتق گیری هستیم. اما در مرتبه دوم مقدار p از ۵٪ کمتر میشود. نتایج و نمودار های مربوط به این تست در ادامه آورده شده:

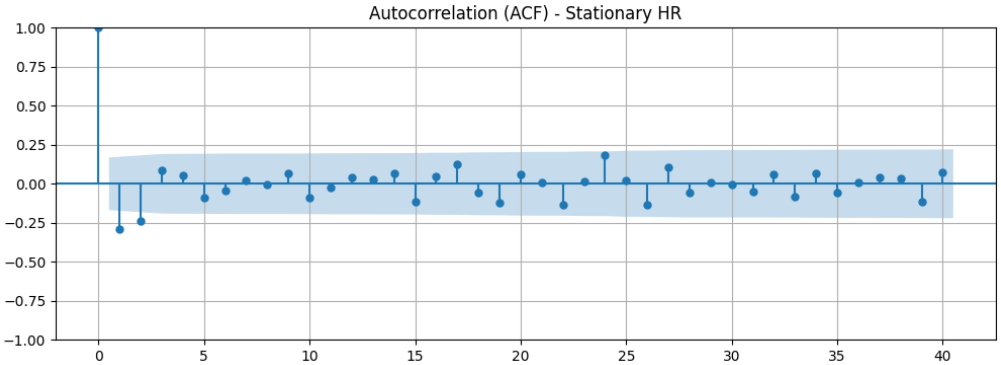


شکل ۳۱ نمودار های **HR** و **O2Sat** بر حسب **Hour**

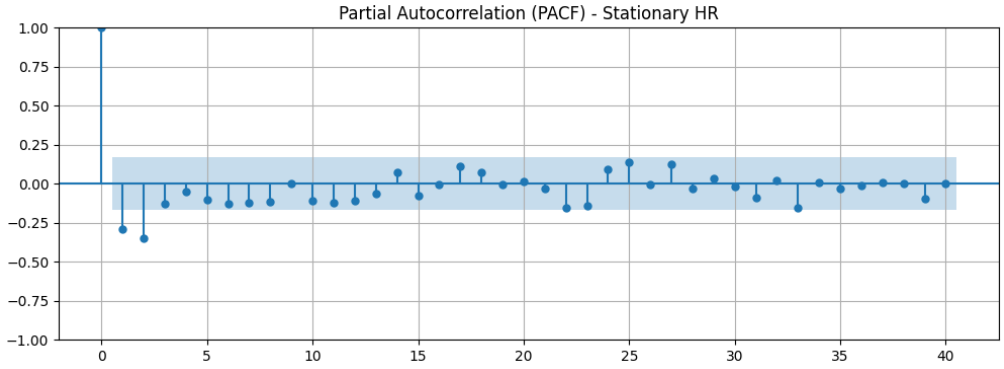


شکل ۳۲ نمودار **HR** برحسب **Hour** پس از یکبار مشتق گیری

بنابراین d=1 است. در ادامه نمودار های Autocorreltaion و Partial AutoCorrelation را برای HR رسم میکنیم. در اولین لگ که کاهش معنا داری دیدیم یعنی AR و MA را پیدا کرده ایم. ولی نکته مهم این است که مقدار لگ اول همواره برابر با ۱ است. بنابراین نباید این لگ را در شمارش های خود حساب کنیم.



شکل ۳۳ نمودار **ACF**



شکل ۳۴ نمودار **PACF**

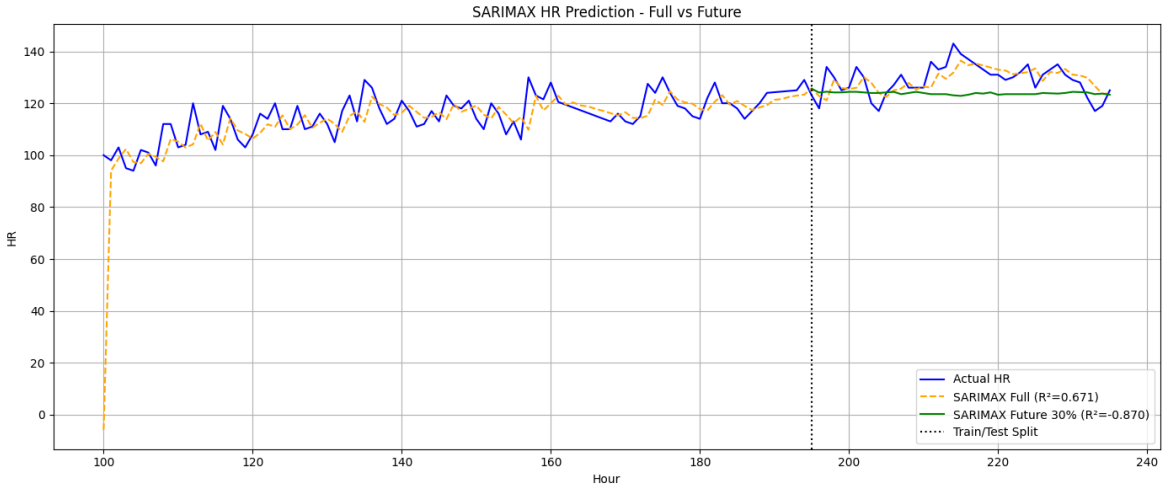
از این نمودار ها واضح است که مقادیر AR و MA برابر با ۲ است زیرا کاهش معنا داره اندازه لگ ها و ورود لگ ها به بازه اطمینان از لگ دوم به بعد است.

با این پارامتر ها SARIMAX را روی داده ای که داشتیم ترین می کنیم. دو مدل روی داده ترین و تست شد:

۱- یک مدل روی کل طول داده ترین و روی کل طول داده تست شد. البته اسم اینکار عملا تست نیست زیرا مدل کل داده را دیده است. عملا می توانیم بگوییم مدل SARIMAX یک بار کل داده را دیده و بار دیگر آن را بازسازی کرده.

۲- یک مدل روی ۷۰٪ ابتدایی طول داده ترین و روی ۳۰٪ باقی مانده تست شده.

نتایج مربوط به هر دو مدل و نمودار پیشبینی شده توسط آنها در زیر آورده شده:



شکل ۳۵ نتیجه **SARIMAX**

```
Final R² scores:
Full model (reconstruction): R² = 0.6715
Split model (future prediction): R² = -0.8700
```

با توجه به نمودار ها بازسازی نمودار با دقت خیلی خوبی انجام شده است و مدل در این مورد موفق بوده. ولی در نمونه ای که مدل روی ۷۰٪ داده ها ترین و روی ۳۰٪ داده ها تست شده مدل اصلا عملکرد خوبی نداشته و R^2 score آن منفی شده. این یعنی اگر مدل هر بار فقط میانگین نمودار را به عنوان خروجی میداد وضعیت بهتر بود. با توجه به اینکه طول داده کم بود، این نتیجه چندان دور از انتظار نیست.

نهایتا داده های ترین را با روش min-max نرمالسازی میکنیم و داد های تست و ولیدیشن را نیز با پارامتر های داده های ترین نرمال سازی میکنیم. ستون ساعت را برای هر مریز نرمال سازی میکنیم و ستون SepsisLabel که داده ای باینری است را نرمالسازی نمیکنیم.

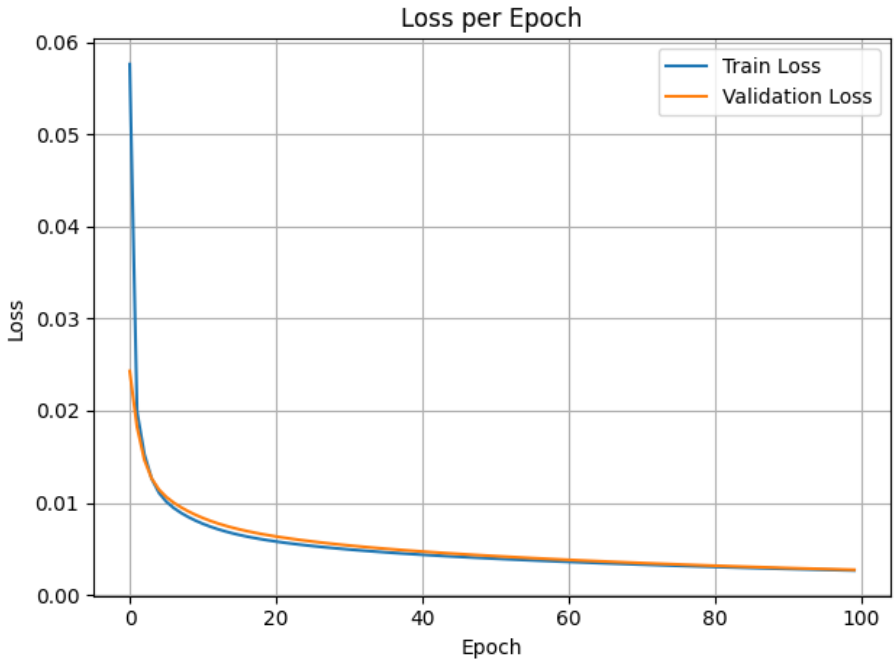
بخش سوم: آموزش مدل های یادگیری عمیق

برای Window خروجی ساده ترین راه این است که این مقدار را برابر با ۱ قرار دهیم. یعنی مدل تعدادی از ورودی ها را ببیند و یک خروجی را پیشبینی کند. ولی برای Window خروجی می توانیم از دو نمودار قبلی مواردی را متوجه شویم:

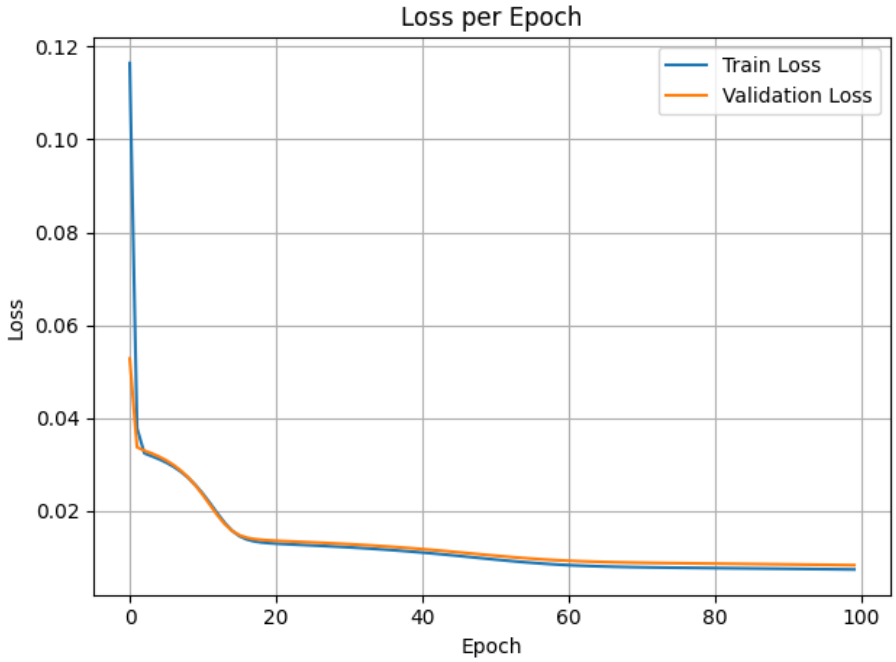
- ACF : این نمودار، همبستگی بین سری زمانی و نسخه‌های تأخیریافته (lagged versions) خودش را نشان می‌دهد. مثلاً همبستگیY_t با Y_{t-1} لگ ۱،Y_{t-1} با Y_{t-2} لگ ۲ و الی آخر. اگر در نمودار ACFهمبستگی‌ها تا مثلاً لگ k به طور معناداری غیر صفر باشند و سپس به سرعت کاهش پیدا کرده و وارد بازه اطمینان شوند، این می‌تواند نشان دهد که مشاهدات تا kگام زمانی گذشته، اطلاعات مفیدی برای پیش‌بینی مشاهده فعلی دارند.
- PACF : این نمودار، همبستگی بین Y_t با Y_{t-k} را پس از حذف اثر همبستگی‌های مربوط به لگ‌های میانی (۱ تا k-1) نشان می‌دهد. در واقع، همبستگی مستقیم بین Y_t و Y_{t-k} را اندازه‌گیری می‌کند. گر نمودار PACF در لگ k به طور ناگهانی قطع شود (یعنی مقادیر بعدی به سرعت وارد بازه اطمینان شوند)، این نشان می‌دهد که پس از در نظر گرفتن وابستگی به k-1 گام زمانی قبلی، وابستگی مستقیم و معناداری به گام‌های دورتر وجود ندارد.

بنابراین مقادیر AR و MA که در بخش قبل بدست آوردیم در این بخش کمک کننده خواهند بود. در بخش قبل برای یک سیگنال و برای یک ویژگی هر دو این مقادیر را 2 بدست آوردیم. اگر این مقدار را برای تمام ستون ها و تمام بیمار ها تعمیم دهیم می توانیم اندازه پنجره ورودی را ۲ درنظر بگیریم ولی محض احتیاط طول پنجره ورودی را ۶ درنظر میگیریم. با این فرض با batch_size = 32 (مطابق با دستور کار) ،input_length = 6 (مطابق با چیزی که بدست آوردیم)، optimizer_type= sgd (مطابق با مقاله) و learning rate = 0.005 (مطابق با مقاله) در ۱۰۰ ایپاک مدل ها را آموزش میدهیم. البته

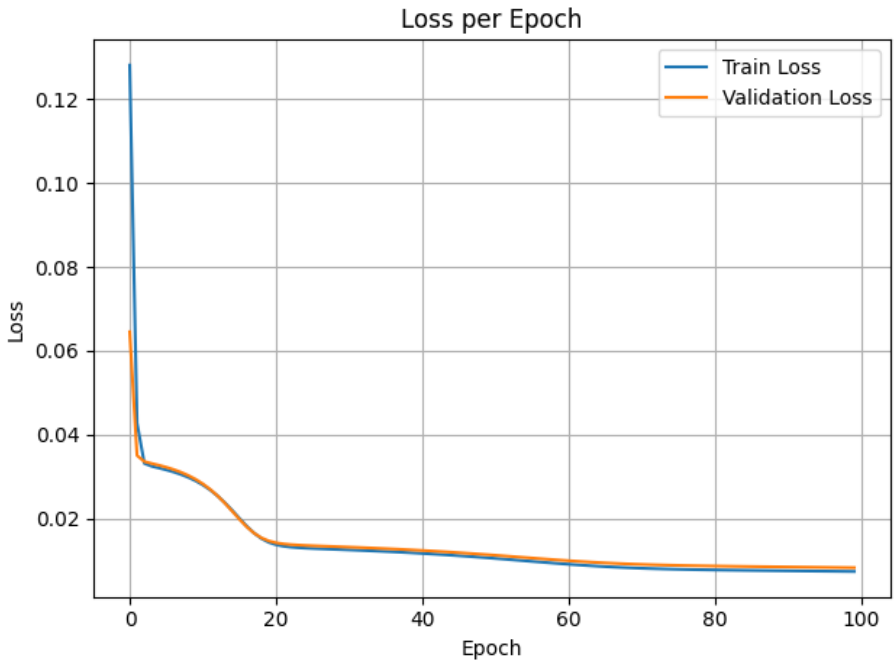
برای مدل ها مکانیزم ارلی استاپ درنظر میگیریم که اگر مدل ها زودتر ترین شدند یادگیری متوقف شود. نمودار های مربوطه در ادامه آمده اند.



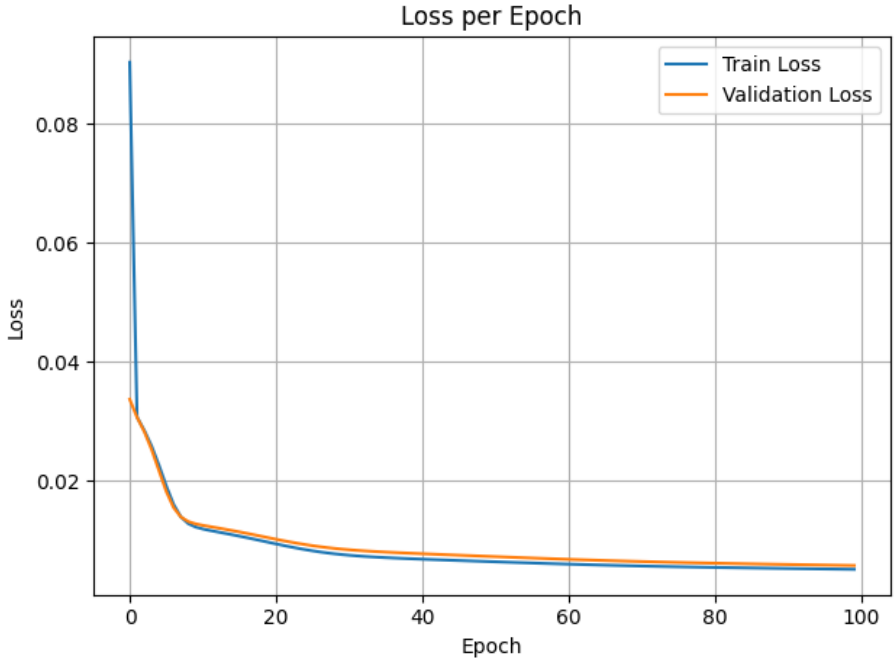
شکل ۳۶ نمودار یادگیری مدل **State-Space Markov Event prediction**



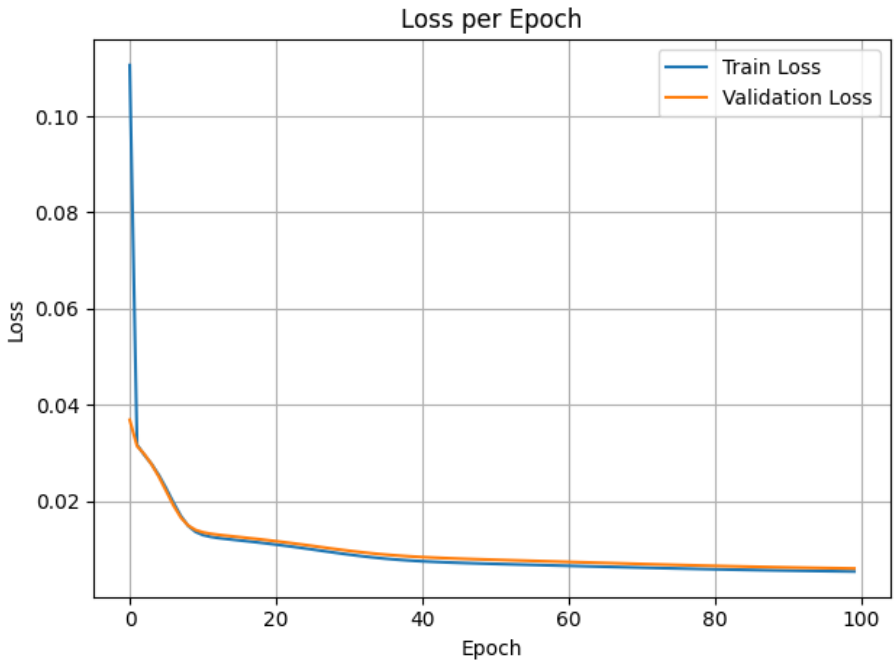
شکل ۳۷ نمودار یادگیری مربوط به **LSTM**



شکل ۳۸ نمودار یادگیری مربوط به **BiLSTM**



شکل ۳۹ نمودار یادگیری مربوط به **GRU**



شکل ۴۰ نمودار یادگیری مربوط به **BiGRU**

نتایج نهایی:

	Loss	MSE	MAE	R2	Cosine
MarkovMLP	0.0376	0.0030	0.0346	-0.4829	0.1568
GRU	0.0544	0.0062	0.0482	-0.8950	0.1449
BiGRU	0.0559	0.0063	0.0495	-1.1335	0.1442
BiLSTM	0.0675	0.0087	0.0588	-0.5596	0.1408
LSTM	0.0676	0.0090	0.0587	-0.3823	0.1405

با توجهه به این نتایج مدل ها اصلا قدرت خوبی ندارند و در اکثر موارد اگه بجای خروجی مدل میانگین سیگنال را قرار دهیم وضعیت بهتر است!

با توجه به اینکه نتایج با تنظیم این هایپر پارامتر ها چندان جالب نشد، مدل ها را با تنظیمات دیگری مجدادا ترین میکنیم:

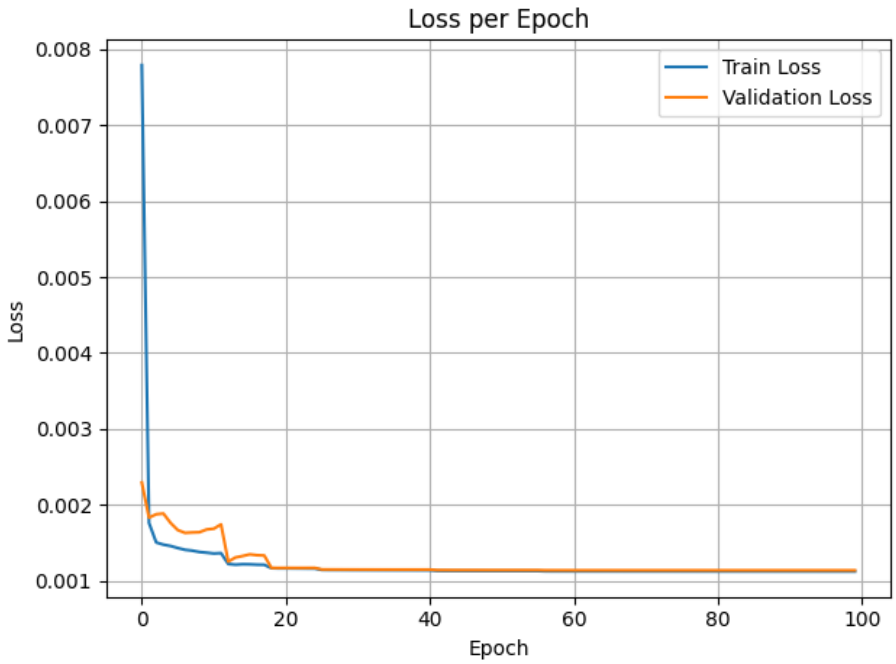
batch_size = 32

input_length = 15

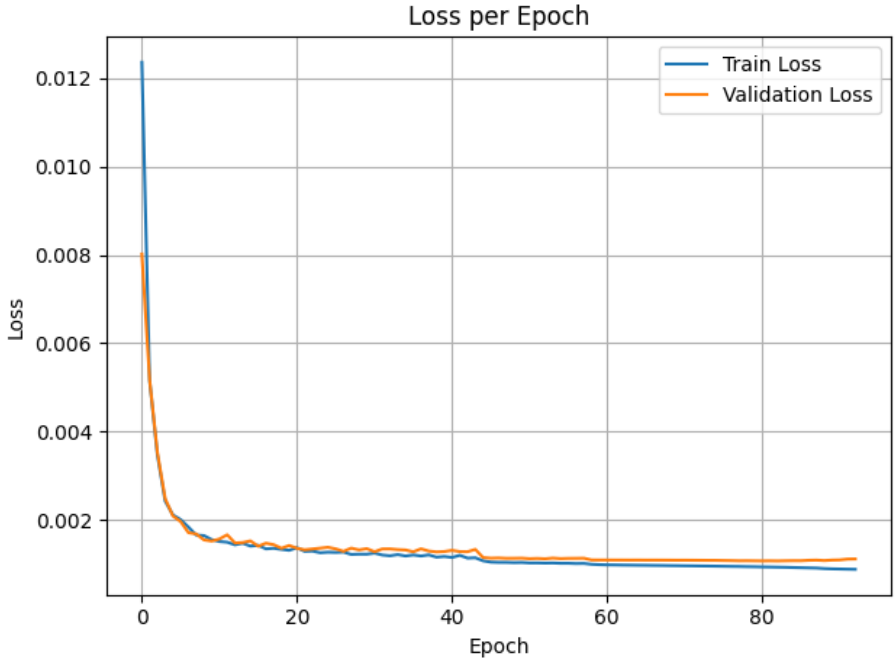
optimizer_type= adam

learning rate متغییر. به این صورت که از ۰.۰۰۱ شروع میشود و بعد از هر ۵ گامی که loss روی داده های ولیدیشن کاهش نیافت، مقدار آن نصف میشود.

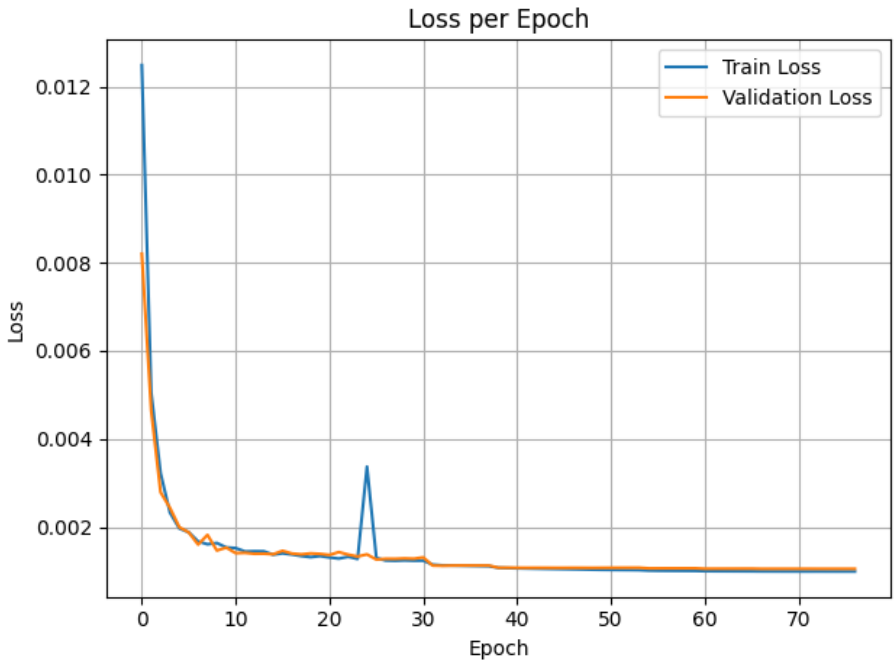
برای این مدل ها هم مکانیزم ارلی استاپ در نظر میگیریم و به مدت ۱۰۰ ایپاک ترین میکنیم. نتایج در ادامه آمده است.



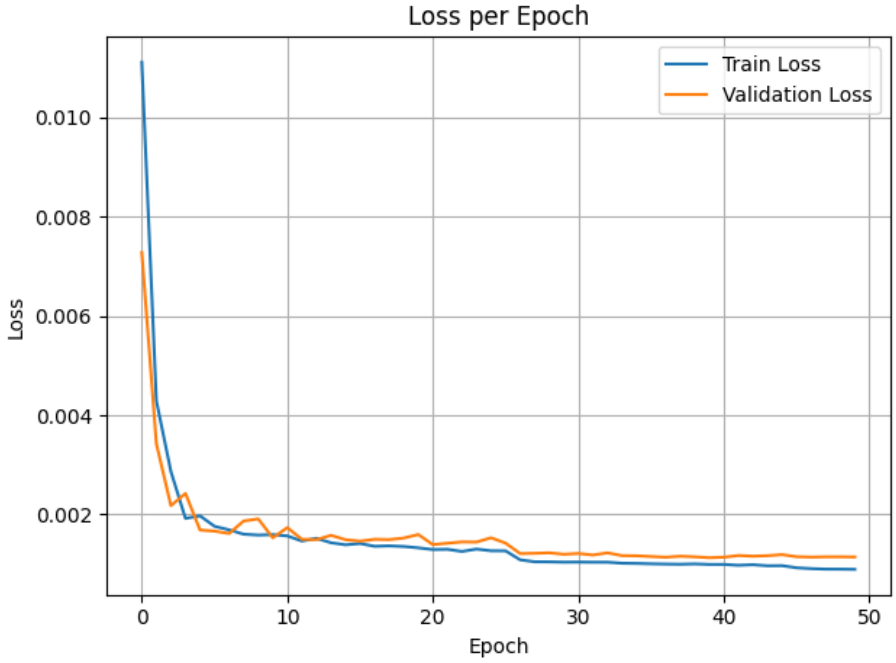
شکل ۴۱ نمودار یادگیری مربوط به **State-Space Markov Event predictio**



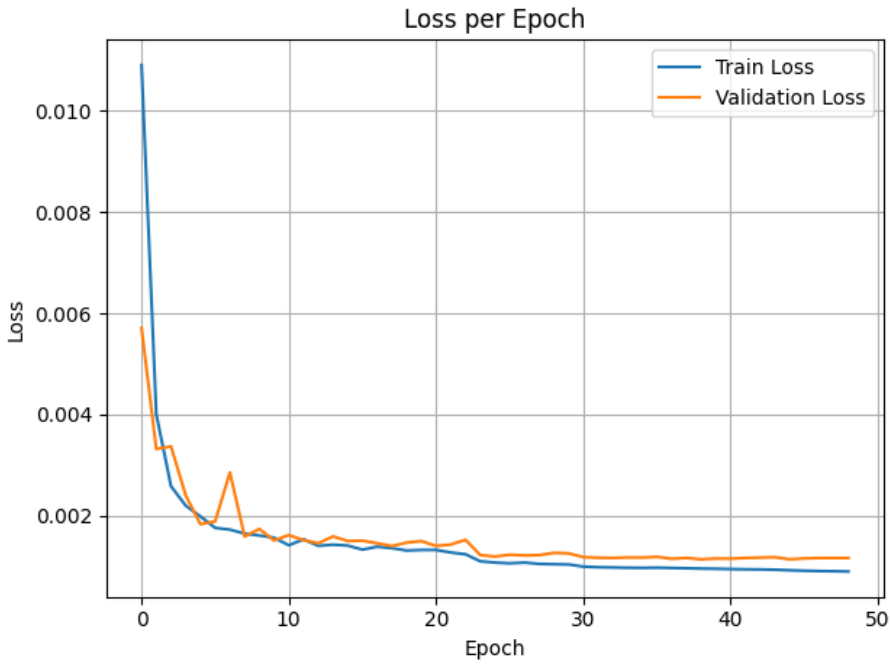
شکل ۴۲ نمودار یادگیری مربوط به **LSTM**



شکل ۴۳نمودار یادگیری مربوط به **BiLSTM**



شکل ۴۴ نمودار یادگیری مربوط به **GRU**



شکل ۴۵ نمودار یادگیری مربوط به **BiGRU**

نتایج نهایی:

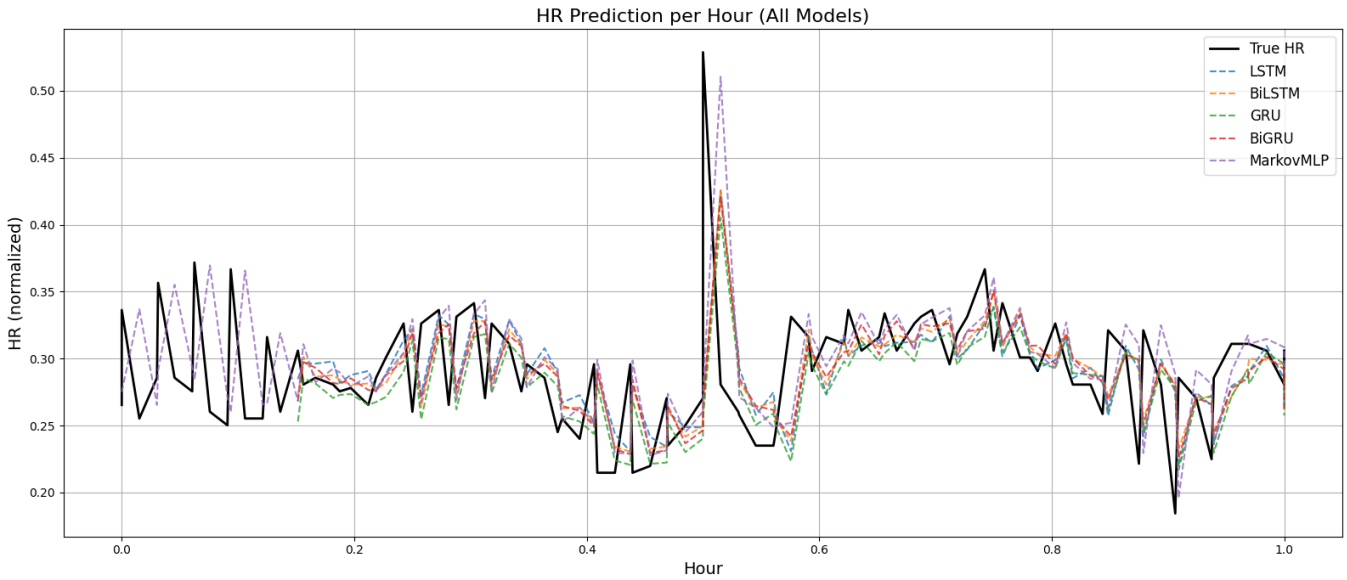
	Loss	MSE	MAE	R2	Cosine
MarkovMLP	0.0141	0.0013	0.0128	0.8624	0.1628
BiLSTM	0.0165	0.0016	0.0149	0.8216	0.1437
GRU	0.0202	0.0019	0.0183	0.7087	0.1455
LSTM	0.0194	0.0020	0.0174	0.7928	0.1444
BiGRU	0.0196	0.0023	0.0173	0.8087	0.1446

با توجه به این نتایج می توان گفت مدل Markov از همه مدل ها بهتر عمل کرده است و نتایج آن بسیار بهتر از پیشبینی میانگین است.

بخش چهارم: رسم نتایج و تحلیل جواب ها

با توجه به اینکه مدل مارکوف در بخش قبل نتیجه بهتری داشته، می توانیم نتیجه بگیریم که ماهیت داده ها به شکلی بوده که نیازی به حافظه عمیق ندارد و می توان از روی داده قبلی، داده لحظه فعلی را پیشبینی کرد. نکته دیگر استفاده از adam و learning rate متغییر است. همانطور که واضح است این دو مورد تاثیر بسیار زیادی روی متریک های ما داشته اند و وضعیت مدل را بهتر کرده اند. نکته دیگر مدل های دوطرفه هستند. همانطور که واضح است و قابل پیشبینی نیز بود، مدل ها دو طرفه نتیجه خیلی بهتری از مدل های یکطرفه دارند. به حدی که BiGRU حدود ده درصد بهتر از GRU عمل کرده است.

در ادامه مقدار HR نرمالایز شده را برحسب Hour نرمالایز شده برای هر ۵ مدل و سیگنال اصلی رسم میکنیم:



شکل ۴۶ نمودار **HR** بر حسب **Hour** برای همه مدل ها و سیگنال اصلی

با توجه به نمودار بالا خروجی مدل ها بسیار نزدیک به سیگنال اصلی بوده است. علت اینکه سیگنال اصلی و مارکوف از ابتداط نمودار وجود دارند و سیگنال خروجی های بقیه مدل ها بعدا به این موارد اضافه میشوند این است که بقیه مدل ها برای پیشبینی به ۱۵ داده نیاز دارند و تا ۱۵ داده ابتدایی را نگیرند نمی توانند خروجی را تولید کنند.

بخش پنجم: روش Maximum Log-Likelihood Estimation

برای آموزش و ارزیابی این مدل، باید در ابتدا داده ها را آماده کنیم. برای آماده سازی این داده ها ابتدا ستون Hour و SepsisLabel را حذف می کنیم زیرا SepsisLabel یک ویژگی باینری است و میانگین و واریانس برای Hour معنی ندارد. سپس با یک پنجره ۹ تایی روی داده ها حرکت میکنیم و میانگین و واریانس در هر لحظه را با استفاده از ۴ داده قبلی و بعدی آن محاسبه میکنیم و در دیتا فریم های جدیدی ذخیره میکنیم. نتیجه فایل های csv ولی با ۳۸ ستون هستند که ۱۹ ستون اول میانگین ها و ۱۹ ستون دوم واریانس ها را نشان میدهند.

علت اینکه جدا کردن میانگین ها و واریانس ها فرض درستی است این است که:

- در اغلب کاربردهای عملی مثل پیش‌بینی ICU یا هزینه‌ی بیماران، فرض استقلال ویژگی‌ها ساده‌سازی مناسبی است.
- این فرضیه منجر به مدل‌های کم‌پارامتر و قابل کنترل می‌شود.

- همچنین، اگر خروجی هر ویژگی به‌صورت مستقل پیش‌بینی شود، در صورت کمبود داده یا نویز، شبکه بهتر آموزش میبیند.

می توان از همان ساختار های قبلی برای این پیشبینی ها استفاده کردی ولی باید یکسری ویژگی ها را تغییر داد:

- ابعاد خروجی و وردی باید تغییر کنند.
- تابع فعال سازی در لایه آخر نیز باید تغییر کنند.

ولی به صورت کلی می توان از همان ساختار ها و معماری ها استفاده کرد.

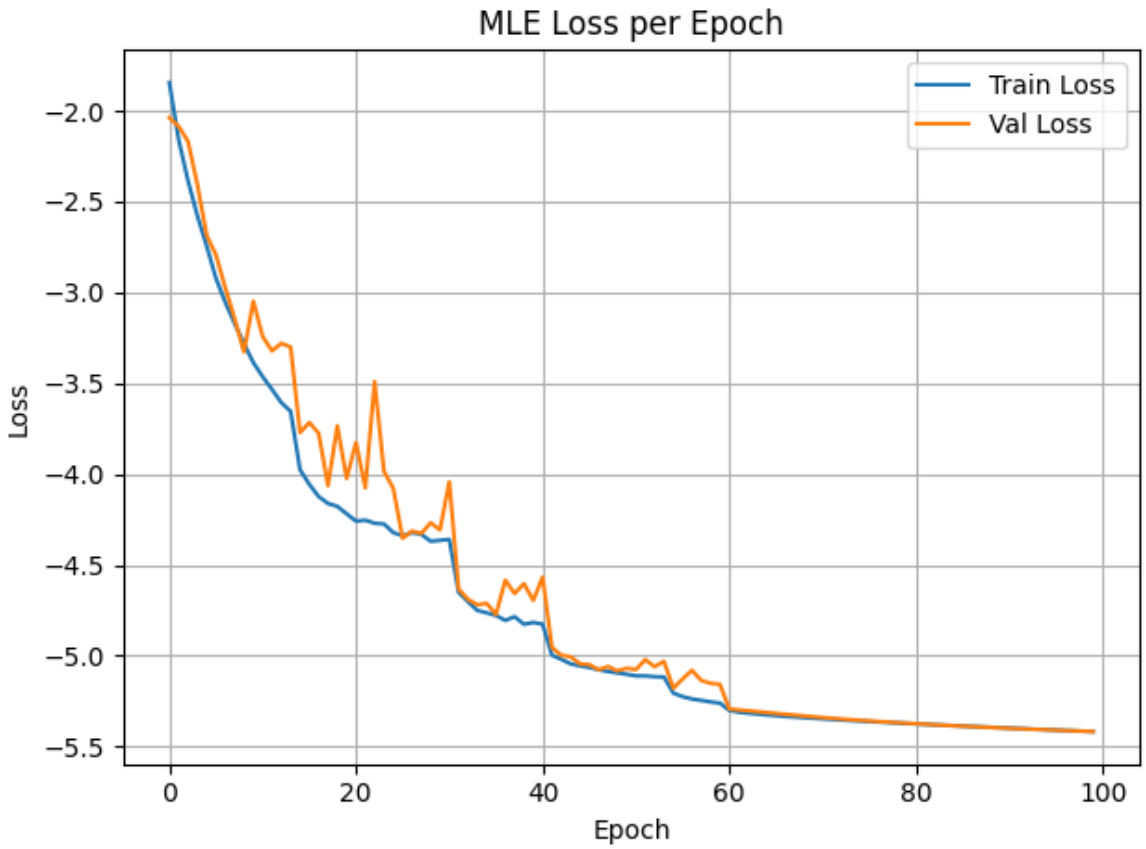
درباره تابع هزینه، با توجه به اینکه می خواهیم توزیع احتمال خروجی را پیشبینی کنیم، بهتر است از Maximum Log-Likelihood Estimation استفاده کنیم.

درباره تابع فعالسازی نیز برای میانگین هیچ محدودیتی نداریم و از اکتیویشن فانکشن استفاده نمیکنیم. ولی درباره واریانس، باید از تابع فعالسازی استفاده کنیم که به عنوان خروجی عدد مثبت دهد. بنابراین می توانیم از exp یا Softplus استفاده کنیم.

نهایتا شبکه را به این شکل طراحی میکنیم:

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
LSTMStatsModel                           [32, 19]                  --
├─LSTM: 1-1                              [32, 15, 128]             86,016
├─Linear: 1-2                            [32, 19]                  2,451
├─Linear: 1-3                            [32, 19]                  2,451
==========================================================================================
Total params: 90,918
Trainable params: 90,918
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 41.44
==========================================================================================
Input size (MB): 0.07
Forward/backward pass size (MB): 0.50
Params size (MB): 0.36
Estimated Total Size (MB): 0.94
==========================================================================================
```

نتایج و نمودار های آموزش شبکه در صفحه بعد آورده شده. نکته جالب این شبکه این است که با توجه به loss function مقدار loss منفی است.



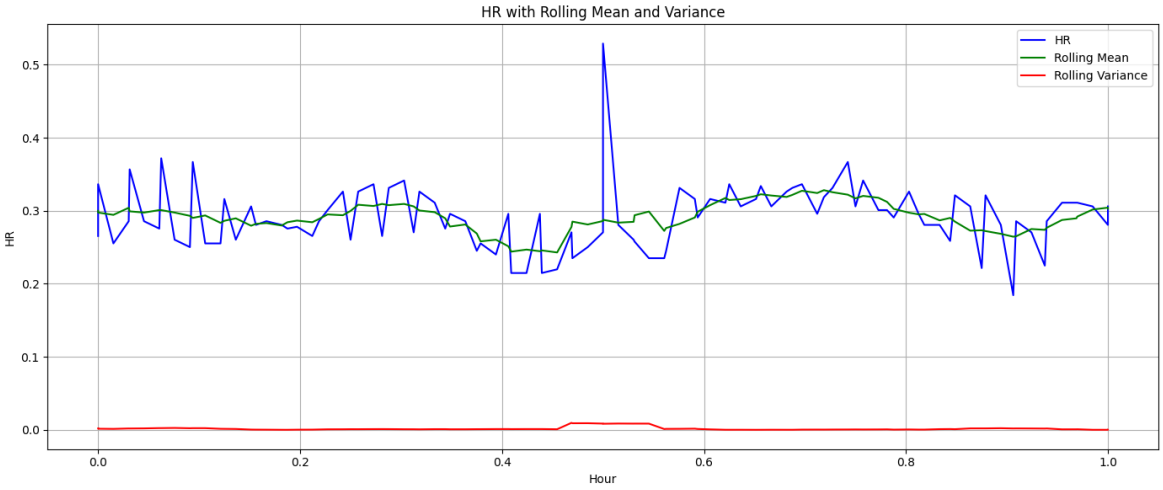
شکل ۴۷ نتایج آموزش شبکه **LSTM** روی داده های میانگین و واریانس

متریک های مورد بررسی روی داده های تست نیز در زیر آورده شده:

```
Evaluation Metrics: {
'Loss': 0.0024219119568442693,
'MSE': 1.6919488189159892e-05,
'MAE': 0.0024049924686551094,
'R2': 0.9518901705741882,
'Cosine': np.float32(0.1377809)}
```

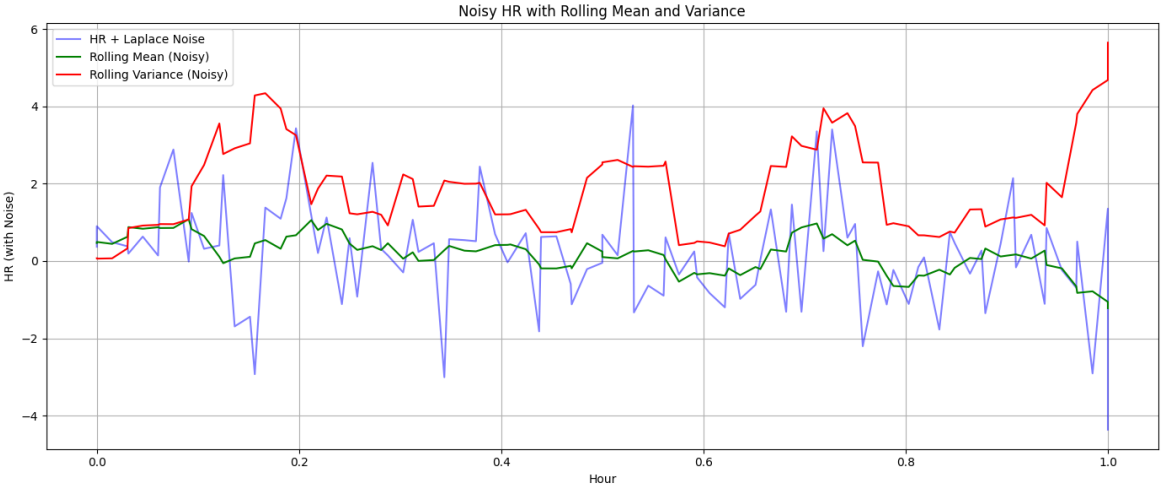
با توجه به این مقادیر مدل بسیار خوب و مناسب آموزش دیده است. دقت کنیم که مقدار loss ای که از تابع MSE یا MAE میگیریم منفی نیست و فقط loss ای که از تابع Maximum Log-Likelihood Estimation میگیریم منفی است.

برای بخش بعدی ابتدا سیگنالی که از انتهای بخش قبل داشتیم و نتایج مدل ها را روی آن نشان دادیم را پلات کرده و مقادیر میانگین و واریانس آن را نیز کنار خودش پلات می کنیم.



شکل ۴۸ سیگنال **HR** و میانگین و واریانس آن

در مرحله بعدی یک نویز روی این سیگنال سوار میکنیم و همین کار ها را تکرار میکنیم.



شکل ۴۹ سیگنال **HR** نویزی همراه با میانگین و واریانس آن

حالا میانگین و واریانس نویزی را به مدل ای که بر اساس میانگین و واریانس ترین کردیم میدهیم و داده های اصلی نویزی را به مدل BiLSTM میدهیم که ببینیم کدام یک عملکرد مقاوم تری در مقابل نویز دارند.