



RNN

practice 인공지능,머신러닝 · 2024. 10. 16. 06:53

RNN

순차적 데이터 (Sequential data)

Gradient vanishing - 로스를 w에 대해 미분했을때 곱하는값이 많아져서 gradient값(기울기 = 미분)이 거의 0된데

Gradient Vectors

$$\frac{\partial y}{\partial w} = \frac{dy}{d\cdot} \cdot \frac{\partial}{\partial \cdot} \cdots \cdots \cdot \frac{\partial}{\partial w}$$

W&H \rightarrow 페인팅

$$y \neq 0 \Rightarrow 0 < 0 \times \dots \times 0 = \dots$$

$$\Rightarrow 0.00000$$

$$w^{n+1} = w^n - \eta \frac{\partial J}{\partial w} \quad \text{gradient vanishing}$$

↳ 홈페이지가 거의 없거나

$$\text{제이슨} \quad z = \frac{2x^2y}{1+y^2} \quad \rightarrow \frac{\partial z}{\partial x} \rightarrow 2y \quad 2x$$

loss

상수취급 \rightarrow

\rightarrow 제이슨 미분

$$\frac{\partial z}{\partial y} = 2x^2$$

Long term dependency

문장 길이가 길어지면 초반 내용을 기억능력이 떨어짐

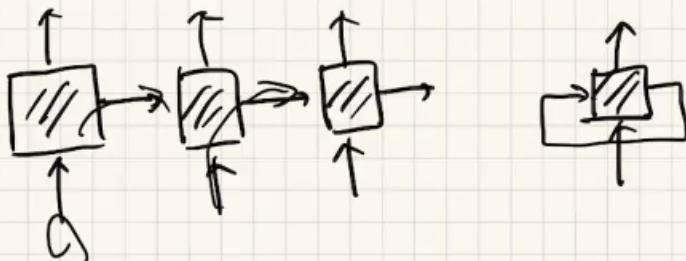
-> 해결책 -> bidirectional RNN -> 좋은 해결책 아님

장기 의존성(Long-term dependencies)은 모델, 시스템 또는 알고리즘이 입력 시퀀스의 초반

부에서 중요한 정보를 시간적 또는 맥락적으로 먼 거리까지 기억하고 활용하는 능력을 말합니

다. 이는 여러 머신러닝 및 자연어 처리(NLP) 작업에서 매우 중요한 개념으로, 특히 문장이나 문서 내에서 멀리 떨어진 단어들이나 개념들 간의 관계를 이해하는 데 필요합니다.

- RNN - 순차적 처리
sequential data

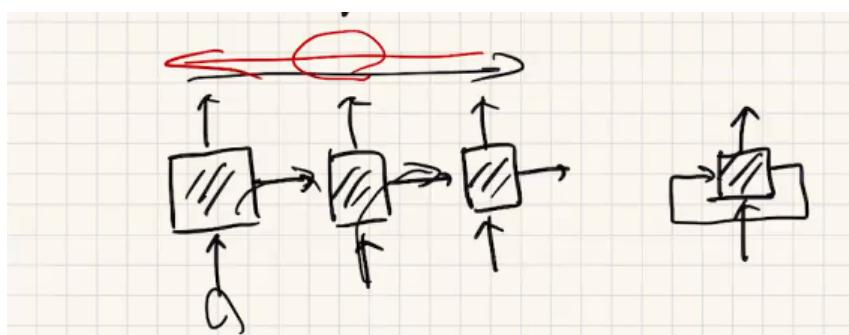


- Gradient vanishing
- long-term dependency

LSMMA , GRU

Attention
↓
Transformer

Bidirectional



6-Surname_Classification_with_RNNs.ipynb

```

if args.reload_from_files and os.path.exists(args.vectorizer_file):
    # 체크포인트를 로드합니다.
    dataset = SurnameDataset.load_dataset_and_load_vectorizer(args.surname_csv,
                                                               args.vectorizer_file)
else:
    # 데이터셋과 Vectorizer를 만듭니다.
    dataset = SurnameDataset.load_dataset_and_make_vectorizer(args.surname_csv)
    dataset.save_vectorizer(args.vectorizer_file)

vectorizer = dataset.get_vectorizer()

classifier = SurnameClassifier(embedding_size=args.char_embedding_size,
                                 num_embeddings=len(vectorizer.char_vocab),
                                 num_classes=len(vectorizer.nationality_vocab),
                                 rnn_hidden_size=args.rnn_hidden_size,
                                 padding_idx=vectorizer.char_vocab.mask_index)

[13] print(dataset._vectorizer.char_vocab._token_to_idx)
[14] print(dataset._vectorizer.nationality_vocab._token_to_idx)

```

6-Surname_Classification_with_RNNs.ipynb

```

# 파일을 열고 데이터를 읽습니다.
surname_df = pd.read_csv(args.surname_csv)
train_surname_df = surname_df[surname_df.split=='train']
split=='train'
return cls(surname_df, SurnameVectorizer.from_dataframe(train_surname_df))

```

6-Surname_Classification_with_RNNs.ipynb

```

def to_serializable(self):
    return {'char_vocab': self.char_vocab.to_serializable(),
            'nationality_vocab': self.nationality_vocab.to_serializable()}

class SurnameDataset(Dataset):
    def __init__(self, surname_df, vectorizer):
        """매개변수:
        surname_df (pandas.DataFrame): 데이터셋
        vectorizer (SurnameVectorizer): 데이터셋에서 만든 Vectorizer 객체
        """
        self.surname_df = surname_df
        self._vectorizer = vectorizer

        self._max_seq_length = max([len(item) for item in self.surname_df['surname']]) + 2
        self.train_df = self.surname_df[self.surname_df.split=='train']
        self.train_size = len(self.train_df)

        self.val_df = self.surname_df[self.surname_df.split=='val']
        self.validation_size = len(self.val_df)

        self.test_df = self.surname_df[self.surname_df.split=='test']
        self.test_size = len(self.test_df)

        self._lookup_dict = {'train': (self.train_df, self.train_size),
                            'val': (self.val_df, self.validation_size),
                            'test': (self.test_df, self.test_size)}

        self.set_split('train')

```

6-Surname_Classification_with_RNNs.ipynb

```

class SurnameDataset(Dataset):
    def __init__(self, surname_df, vectorizer):
        """매개변수:
        surname_df (str): 데이터셋의 위치
        반환값:
        SurnameDataset의 객체
        """
        surname_df = pd.read_csv(surname_csv)
        train_surname_df = surname_df[surname_df.split=='train']
        split=='train'
        return cls(surname_df, SurnameVectorizer.from_dataframe(train_surname_df))

```

The screenshot shows a dual-monitor setup. The left monitor displays a Jupyter Notebook cell containing Python code for a surname classification model. The right monitor displays a code editor with the same code, with several annotations in red and yellow highlighting specific parts of the code, likely indicating areas of interest or modification.

```

124 class SurnameVectorizer(object):
125     """이름 벡터화를 생성하고 관리합니다 """
126     def __init__(self, char_vocab, nationality_vocab):
127         """
128             매개변수:
129                 char_vocab (Vocabulary): 문자를 정수로 매핑합니다
130                 nationality_vocab (Vocabulary): 국적을 정수로 매핑합니다
131         """
132         self.char_vocab = char_vocab
133         self.nationality_vocab = nationality_vocab
134
135     def vectorize(self, surname, vector_length=1):
136         """
137             매개변수:
138                 title (str): 문자열
139                 vector_length (int): 인덱스 벡터의 길이를 맞추기 위한 매개변수
140         """
141         indices = [self.char_vocab.begin_seq_index]
142         indices.extend(self.char_vocab.lookup_token(token)
143                         for token in surname) # surname을 문자 하나하나로 반복
144         indices.append(self.char_vocab.end_seq_index) # lookup token으로 index를 찾는다
145         if vector_length < 0:
146             vector_length = len(indices) # 각 문자가 몇 번째 token인지 찾음
147         out_vector = np.zeros(vector_length, dtype=np.int64)
148         out_vector[:len(indices)] = indices
149         out_vector[len(indices):] = self.char_vocab.mask_index
150
151     return out_vector, len(indices)
152
153 @classmethod
154 def from_dataframe(cls, surname_df):
155     """데이터셋 데이터프레임으로 SurnameVectorizer 객체를 초기화합니다."""

```

This screenshot shows a similar dual-monitor setup. The left monitor displays a Jupyter Notebook cell with code for printing dataset vectors. The right monitor displays the corresponding code in a code editor, with red and yellow annotations highlighting specific parts of the code, particularly the vectorization logic.

```

124 class SurnameVectorizer(object):
125     """이름 벡터화를 생성하고 관리합니다 """
126     def __init__(self, char_vocab, nationality_vocab):
127         """
128             매개변수:
129                 char_vocab (Vocabulary): 문자를 정수로 매핑합니다
130                 nationality_vocab (Vocabulary): 국적을 정수로 매핑합니다
131         """
132         self.char_vocab = char_vocab
133         self.nationality_vocab = nationality_vocab
134
135     def vectorize(self, surname, vector_length=1):
136         """
137             매개변수:
138                 title (str): 문자열
139                 vector_length (int): 인덱스 벡터의 길이를 맞추기 위한 매개변수
140         """
141         indices = [self.char_vocab.begin_seq_index]
142         indices.extend(self.char_vocab.lookup_token(token)
143                         for token in surname) # surname을 문자 하나하나로 반복
144         indices.append(self.char_vocab.end_seq_index) # lookup token으로 index를 찾는다
145         if vector_length < 0:
146             vector_length = len(indices) # 각 문자가 몇 번째 token인지 찾음
147         out_vector = np.zeros(vector_length, dtype=np.int64)
148         out_vector[:len(indices)] = indices
149         out_vector[len(indices):] = self.char_vocab.mask_index
150
151     return out_vector, len(indices)
152
153 @classmethod
154 def from_dataframe(cls, surname_df):
155     """데이터셋 데이터프레임으로 SurnameVectorizer 객체를 초기화합니다."""

```

The screenshot shows a dual-monitor setup. The left monitor displays a Jupyter Notebook interface with code cells and output. The right monitor displays PyCharm with an open file named '6-Surname_Classification_with_RNNs.ipynb'. A red arrow points from the Jupyter notebook's output area to the PyCharm code editor, highlighting the line 'def getitem_(self, index):'.

```
185 class SurnameDataset(Dataset):
186     def set_split(self, split='train'):
187         self._target_df, self._target_size = self._lookup_dict[split]
188
189     def __len__(self):
190         return self._target_size
191
192     def getitem_(self, index):
193         """파이썬 데이터셋의 주요 접근 메서드
194
195         매개변수:
196             index (int): 데이터 포인트 인덱스
197
198         반환값:
199             다음 값을 담고 있는 딕셔너리:
200                 특성 (x_data)
201                 레이블 (y_target)
202                 특성 길이 (x_length)
203
204         row = self._target_df.iloc[index]
205
206         surname_vector, vec_length = \
207             self._vectorizer.vectorize(row.surname, self._max_seq_length)
208
209         nationality_index = \
210             self._vectorizer.nationality_vocab.lookup_token(row.nationality)
211
212         return {'x_data': surname_vector,
213                 'y_target': nationality_index,
214                 'x_length': vec_length}
215
216     def get_num_batches(self, batch_size):
217         """배치 크기가 주어지면 데이터셋으로 만들 수 있는 배치 개수를
218         반환합니다
219
220         매개변수:
221             batch_size (int): 배치 크기
222
223         반환값:
224             int: 데이터셋으로 만들 수 있는 배치 개수
225
226         """
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
```

연속형 데이터는 순서에 따라 단위의 의미가 달라지는 데이터이다. 상당사의 질의응답, 한국어-영어 번역(병행) 말뭉치나 도서자료 요약 등을 일반적으로 순서에 따라 데이터의 의미가 달라져버린다. 반면 일반적인 사진 등은 순서를 바꾸어도 내용이 크게 달라지지 않는다.

RNN의 구조

등호의 왼쪽은 RNN의 전체적인 구조이다.
전체적인 구조의 3개의 화살표를 나눠보면 아래와 같다.

- 입력 벡터가 은닉층에 들어가는 것을 나타내는 화살표
- 은닉층으로부터 출력 벡터가 생성되는 것을 나타내는 화살표
- 은닉층에서 나와 다시 은닉층으로 입력되는 것을 나타내는 화살표

기존 신경망에서는 은닉층에서 나와 다시 은닉층으로 입력되는 과정은 없었다.
이 화살표는 특정 시점에서의 은닉 벡터가 다음 시점의 입력 벡터로 다시 들어가는 과정을 나타내고 있다.

내 컴퓨터를 guys07@gmail.com과 공유하고 있습니다. [공유 중지]

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

문자 하나하나가 $x_1, x_2 \dots$ 자리에 들어감

def forward(self, x_in, x_lengths=None, apply_softmax=False):
 """ 분류기의 정방향 계산
 예개념수:
 x_in (torch.Tensor): 입력 데이터 텐서
 x_in.shape는 (batch, input_dim)입니다
 x_lengths (torch.Tensor): 배치에 있는 각 시퀀스의 길이
 시퀀스의 마지막 벡터를 찾는 데 사용합니다
 apply_softmax (bool): 소프트맥스 활성화 함수를 위한 플래그
 크로스-엔트로피 손실을 사용하려면 False로 지정합니다
 변환값:
 결과 텐서. tensor.shape는 (batch, output_dim)입니다
 ...
 x_embedded = self.emb(x_in) → RNN
 y_out = self.rnn(x_embedded) → embedding해서 vector화해서
 if x_lengths is not None:
 y_out = column_gather(y_out, x_lengths)
 else:
 y_out = y_out[:, -1, :]
 # y_out -> (b, h)
 y_out = F.relu(self.fc1(F.dropout(y_out, 0.5))) # (b, h)
 y_out = self.fc2(F.dropout(y_out, 0.5)) # (b, num_classes)
 if apply_softmax:
 y_out = F.softmax(y_out, dim=1)

연속형 데이터는 순서에 따라 단위의 의미가 달라지는 데이터이다. 상당사의 질의응답, 한국어-영어 번역(병행) 말뭉치나 도서자료 요약 등을 일반적으로 순서에 따라 데이터의 의미가 달라져버린다. 반면 일반적인 사진 등은 순서를 바꾸어도 내용이 크게 달라지지 않는다.

RNN의 구조

등호의 왼쪽은 RNN의 전체적인 구조이다.
전체적인 구조의 3개의 화살표를 나눠보면 아래와 같다.

- 입력 벡터가 은닉층에 들어가는 것을 나타내는 화살표
- 은닉층으로부터 출력 벡터가 생성되는 것을 나타내는 화살표
- 은닉층에서 나와 다시 은닉층으로 입력되는 것을 나타내는 화살표

기존 신경망에서는 은닉층에서 나와 다시 은닉층으로 입력되는 과정은 없었다.
이 화살표는 특정 시점에서의 은닉 벡터가 다음 시점의 입력 벡터로 다시 들어가는 과정을 나타내고 있다.

내 컴퓨터를 guys07@gmail.com과 공유하고 있습니다. [공유 중지]

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

x_lengths

def forward(self, x_in, x_lengths=None, apply_softmax=False):
 """ 분류기의 정방향 계산
 예개념수:
 x_in (torch.Tensor): 입력 데이터 텐서
 x_in.shape는 (batch, input_dim)입니다
 x_lengths (torch.Tensor): 배치에 있는 각 시퀀스의 길이
 시퀀스의 마지막 벡터를 찾는 데 사용합니다
 apply_softmax (bool): 소프트맥스 활성화 함수를 위한 플래그
 크로스-엔트로피 손실을 사용하려면 False로 지정합니다
 변환값:
 결과 텐서. tensor.shape는 (batch, output_dim)입니다
 ...
 x_embedded = self.emb(x_in) → RNN
 y_out = self.rnn(x_embedded) → embedding해서 vector화해서
 if x_lengths is not None:
 y_out = column_gather(y_out, x_lengths)
 else:
 y_out = y_out[:, -1, :]
 # y_out -> (b, h)
 y_out = F.relu(self.fc1(F.dropout(y_out, 0.5))) # (b, h)
 y_out = self.fc2(F.dropout(y_out, 0.5)) # (b, num_classes)
 if apply_softmax:
 y_out = F.softmax(y_out, dim=1)

연속형 데이터는 순서에 따라 단위의 의미가 달라지는 데이터이다. 상당사의 질의응답, 한국어-영어 번역(병렬) 말뭉치나 도서자료 요약 등을 일반적으로 순서에 따라 데이터의 의미가 달라져버린다. 반면 일반적인 사진 등은 순서를 바꾸어도 내용이 크게 달라지지 않는다.

RNN의 구조

등호의 원쪽은 RNN의 전체적인 구조이다.
전체적인 구조의 3개의 화살표를 나눠보면 아래와 같다.

- 입력 벡터가 은닉층에 들어가는 것을 나타내는 화살표
- 은닉층으로부터 출력 벡터가 생성되는 것을 나타내는 화살표
- 은닉층에서 나와 다시 은닉층으로 입력되는 것을 나타내는 화살표

기존 신경망에서는 은닉층에서 나와 다시 은닉층으로 입력되는 과정은 없었다.
이 화살표는 특정 시점에서의 은닉 벡터가 다음 시점의 입력 벡터로 다시 들어가는 과정을 나타내고 있다.

```

417 class SurnameClassifier(nn.Module):
418     def __init__(self, embedding_size, num_embeddings, num_classes,
419                  hidden_size=rnn.hidden_size,
420                  padding_idx=rnn.padding_idx,
421                  batch_first=batch_first):
422         super().__init__()
423         self.embedding = nn.Embedding(num_embeddings, embedding_size)
424         self.rnn = ElmanRNN(input_size=embedding_size,
425                             hidden_size=hidden_size,
426                             batch_first=batch_first)
427         self.fc1 = nn.Linear(in_features=rnn.hidden_size,
428                             out_features=rnn.hidden_size)
429         self.fc2 = nn.Linear(in_features=rnn.hidden_size,
430                             out_features=num_classes)
431
432     def forward(self, x_in, x_lengths=None, apply_softmax=False):
433         """ 분류기의 정방향 계산 """
434
435         x_in = self.embedding(x_in)
436
437         h, _ = self.rnn(x_in)
438
439         h = self.fc1(h)
440
441         if apply_softmax:
442             h = self.fc2(h)
443         else:
444             h = h[:, -1]
445
446         if x_lengths is not None:
447             h = h[torch.arange(h.size(0)), x_lengths - 1]
448
449         return h
    
```

연속형 데이터는 순서에 따라 단위의 의미가 달라지는 데이터이다. 상당사의 질의응답, 한국어-영어 번역(병렬) 말뭉치나 도서자료 요약 등을 일반적으로 순서에 따라 데이터의 의미가 달라져버린다. 반면 일반적인 사진 등은 순서를 바꾸어도 내용이 크게 달라지지 않는다.

RNN의 구조

등호의 원쪽은 RNN의 전체적인 구조이다.
전체적인 구조의 3개의 화살표를 나눠보면 아래와 같다.

- 입력 벡터가 은닉층에 들어가는 것을 나타내는 화살표
- 은닉층으로부터 출력 벡터가 생성되는 것을 나타내는 화살표
- 은닉층에서 나와 다시 은닉층으로 입력되는 것을 나타내는 화살표

기존 신경망에서는 은닉층에서 나와 다시 은닉층으로 입력되는 과정은 없었다.
이 화살표는 특정 시점에서의 은닉 벡터가 다음 시점의 입력 벡터로 다시 들어가는 과정을 나타내고 있다.

```

356 class ElmanRNN(nn.Module):
357     def __init__(self, input_size, hidden_size, batch_first=False):
358         super().__init__()
359         self.batch_first = batch_first
360         self.hidden_size = hidden_size
361
362     def _initial_hidden(self, batch_size):
363         return torch.zeros((batch_size, self.hidden_size))
364
365     def forward(self, x_in, initial_hidden=None):
366         """ ElmanRNN의 정방향 계산 """
367
368         매개변수:
369         x_in (torch.Tensor): 입력 데이터 텐서
370         If self.batch_first: x_in.shape = (batch_size,
371                                         seq_size, feat_size) 단이의길이 batch사이즈 vector사이즈(길이)
372         Else: x_in.shape = (seq_size, batch_size, feat_size)
373         initial_hidden (torch.Tensor): RNN의 초기 은닉 상태
374         변환값:
375         hiddens (torch.Tensor): 각 타임 스텝에서 RNN 출력
376         If self.batch_first:
377             hiddens.shape = (batch_size, seq_size, hidden_size)
378         Else: hiddens.shape = (seq_size, batch_size,
379                               hidden_size)
380
381         if self.batch_first:
382             batch_size, seq_size, feat_size = x_in.size()
383             x_in = x_in.permute(1, 0, 2)
384         else:
385             seq_size, batch_size, feat_size = x_in.size()
386
387             # x_in: (s, b, f)
388
389             hiddens = []
390
391             if initial_hidden is None:
392                 initial_hidden = self._initial_h(
393                     batch_size, self.hidden_size)
394
395             # initial_hidden: (b, s, f)
396             initial_hidden = initial_hidden.permute(1, 0, 2)
397
398             hiddens.append(initial_hidden)
399
400             for i in range(seq_size):
401                 if self.batch_first:
402                     hiddens.append(self._step(x_in[i], hiddens[-1],
403                                             initial_hidden))
404                 else:
405                     hiddens.append(self._step(x_in[i], hiddens[-1],
406                                             initial_hidden))
407
408             if self.batch_first:
409                 hiddens = torch.stack(hiddens).permute(1, 0, 2)
410             else:
411                 hiddens = torch.stack(hiddens)
412
413             return hiddens
    
```

연속형 데이터는 순서에 따라 단위의 의미가 달라지는 데이터이다. 상당사의 질의응답, 한국어-영어 번역(병렬) 말풍자나 도서자료 요약 등은 일반적으로 순서에 따라 데이터의 의미가 달라져버린다. 반면 일반적인 사진 등은 순서를 바꾸어도 내용이 크게 달라지지 않는다.

RNN의 구조

등호의 왼쪽은 RNN의 전체적인 구조이다.
전체적인 구조의 3개의 화살표를 나눠보면 아래와 같다.

- 입력 벡터가 은닉층에 들어가는 것을 나타내는 화살표
- 은닉층으로부터 출력 벡터가 생성되는 것을 나타내는 화살표
- 은닉층에서 나와 다시 은닉층으로 입력되는 것을 나타내는 화살표

기존 신경망에서는 은닉층에서 나와 다시 은닉층으로 입력이 화살표는 특정 시점에서의 은닉 벡터가 다음 시점의 입/나타내고 있다.

```

356 class ElmanRNN(nn.Module):
375     def forward(self, x_in, initial_hidden=None):
376         if self.batch_first:
377             hiddens.shape = (batch_size, seq_size, hidden_size)
378         else: hiddens.shape = (seq_size, batch_size,
379             hidden_size)
380         ...
381         if self.batch_first:
382             batch_size, seq_size, feat_size = x_in.size()
383             x_in = x_in.permute(1, 0, 2)
384         else:
385             seq_size, batch_size, feat_size = x_in.size()
386             ...
387             # x_in: (s, b, f) x_in은 sequence * batch * feature size를 금하는 것 같다.
388             x_in[4] 한정 b*f
389             hiddens = []
390             if initial_hidden is None:
391                 initial_hidden = self._initial_hidden(batch_size)
392                 initial_hidden = initial_hidden.to(x_in.device)
393             hidden_t = initial_hidden
394             ...
395             for t in range(seq_size):
396                 hidden_t = self.rnn_cell(x_in[t], hidden_t) # x_in[t]: (b, f), hidden_t: (b, h)
397                 hiddens.append(hidden_t)
398             hiddens = torch.stack(hiddens) # (s, b, h)
399             if self.batch_first:
400                 hiddens = hiddens.permute(1, 0, 2)
401             ...
402             return hiddens

```

3행 2열이 4개 있다

```

[x]  [37] import numpy as np
      x = np.arange(1, 25).reshape((4, 3, 2))
      print(x)
[[[ 1  2]
  [ 3  4]
  [ 5  6]
  [ 7  8]
  [ 9 10]
  [11 12]
  [[13 14]
  [15 16]
  [17 18]]
  [[19 20]
  [21 22]
  [23 24]]]

```

```

356 class ElmanRNN(nn.Module):
375     def forward(self, x_in, initial_hidden=None):
376         if self.batch_first:
377             hiddens.shape = (batch_size, seq_size, hidden_size)
378         else: hiddens.shape = (seq_size, batch_size,
379             hidden_size)
380         ...
381         if self.batch_first:
382             batch_size, seq_size, feat_size = x_in.size()
383             x_in = x_in.permute(1, 0, 2)
384         else:
385             seq_size, batch_size, feat_size = x_in.size()
386             ...
387             # x_in: (s, b, f)
388             hiddens = []
389             if initial_hidden is None:
390                 initial_hidden = self._initial_hidden(batch_size)
391                 initial_hidden = initial_hidden.to(x_in.device)
392             hidden_t = initial_hidden
393             ...
394             for t in range(seq_size):
395                 hidden_t = self.rnn_cell(x_in[t], hidden_t) # x_in[t]: (b, f), hidden_t: (b, h)
396                 hiddens.append(hidden_t)
397             hiddens = torch.stack(hiddens) # (s, b, h)
398             if self.batch_first:
399                 hiddens = hiddens.permute(1, 0, 2)
400             ...
401             return hiddens

```

```

356 class ElmanRNN(nn.Module):
357     def forward(self, x_in, initial_hidden=None):
358         if initial_hidden is None:
359             hiddens = []
360         else:
361             hiddens.shape = (batch_size, seq_size, hidden_size)
362
363         if self.batch_first:
364             batch_size, seq_size, feat_size = x_in.size()
365             x_in = x_in.permute(1, 0, 2)
366         else:
367             seq_size, batch_size, feat_size = x_in.size()
368
369         # x_in: (s, b, f)
370
371         hiddens = []
372
373         if initial_hidden is None:
374             initial_hidden = self._initial_hidden(batch_size)
375             initial_hidden = initial_hidden.to(x_in.device)
376
377         hidden_t = initial_hidden
378
379         for t in range(seq_size):
380             hidden_t = self.rnn_cell(x_in[t], hidden_t) # x_in[t]: (b, f), hidden_t: (b, h)
381             hiddens.append(hidden_t)
382
383         hiddens = torch.stack(hiddens) # (s, b, h)
384
385         if self.batch_first:
386             hiddens = hiddens.permute(1, 0, 2)
387
388         return hiddens
389
390
391 class SurnameClassifier(nn.Module):
392     def forward(self, x):
393         pass

```

예) 글만식인 서인

연속형 데이터는 순서에 따라 단위의 의미가 달라지는 데이터이다. 상담사의 질의응답, 한국어-영어 번역(병렬) 말뭉치나 도서자료 요약 등을 일반적으로 순서에 따라 데이터의 의미가 달라져버린다. 반면 일반적인 사진 등은 순서를 바꾸어도 내용이 크게 달라지지 않는다.

RNN의 구조

등호의 왼쪽은 RNN의 전체적인 구조이다.
전체적인 구조의 3개의 화살표를 나눠보면 아래와 같다.

- 입력 벡터가 은닉층에 들어가는 것을 나타내는 화살표
- 은닉층으로부터 출력 벡터가 생성되는 것을 나타내는 화살표
- 은닉층에서 나와 다시 은닉층으로 입력되는 것을 나타내는 화살표

기존 신경망에서는 은닉층에서 나와 다시 은닉층으로 입력되는 과정은 없었다.
이 화살표는 특정 시점에서의 은닉 벡터가 다음 시점의 입력 벡터로 다시 들어가는 과정을 나타내고 있다.

```

356 class ElmanRNN(nn.Module):
357     def forward(self, x_in, initial_hidden=None):
358         if initial_hidden is None:
359             hiddens = []
360         else:
361             hiddens.shape = (batch_size, seq_size, hidden_size)
362
363         if self.batch_first:
364             batch_size, seq_size, feat_size = x_in.size()
365             x_in = x_in.permute(1, 0, 2)
366         else:
367             seq_size, batch_size, feat_size = x_in.size()
368
369         # x_in: (s, b, f)
370
371         hiddens = []
372
373         if initial_hidden is None:
374             initial_hidden = self._initial_hidden(batch_size)
375             initial_hidden = initial_hidden.to(x_in.device)
376
377         hidden_t = initial_hidden
378
379         for t in range(seq_size):
380             hidden_t = self.rnn_cell(x_in[t], hidden_t) # x_in[t]: (b, f), hidden_t: (b, h)
381             hiddens.append(hidden_t)
382
383         hiddens = torch.stack(hiddens) # (s, b, h)
384
385         if self.batch_first:
386             hiddens = hiddens.permute(1, 0, 2)
387
388         return hiddens
389
390
391 class SurnameClassifier(nn.Module):
392     def forward(self, x):
393         pass

```

The screenshot shows a dual-monitor setup. The left monitor displays a Jupyter Notebook interface with a code cell for predicting nationality based on a surname. The right monitor displays a PyCharm editor with the same code, annotated with Korean comments explaining the code logic.

```

417 class SurnameClassifier(nn.Module):
418     def __init__(self, embedding_size, num_embeddings, num_classes,
419                  in_features=rnn_hidden_size,
420                  out_features=rnn_hidden_size):
421         super(SurnameClassifier, self).__init__()
422         self.fc1 = nn.Linear(in_features=rnn_hidden_size,
423                             out_features=rnn_hidden_size)
424         self.fc2 = nn.Linear(in_features=rnn_hidden_size,
425                             out_features=num_classes)
426
427     def forward(self, x_in, x_lengths=None, apply_softmax=False):
428         """ 분류기의 정방향 계산
429
430         예제 변수:
431             x_in (torch.Tensor): 입력 데이터 텐서
432             > in.shape는 (batch, input dim)입니다.
433             x_lengths (torch.Tensor): 배치에 있는 각 시퀀스의 길이
434             [원스의 마지막 벤더를 찾는데 사용합니다]
435             apply_softmax (bool): 소프트맥스 활성화 함수를 위한 플래그
436             [로스-엔트로피 손실을 사용하려면 False로 지정합니다]
437
438         반환 값:
439             결과 텐서. tensor.shape는 (batch, output_dim)입니다.
440             ...
441
442             x_embedded = self.embedding(x_in)
443             y_out = self.rnn(x_embedded)
444
445             if x_lengths is not None:
446                 y_out = column_gather(y_out, x_lengths)
447             else:
448                 y_out = y_out[:, -1, :]
449             # y_out -> (b, h)
450             y_out = F.relu(self.fc1(F.dropout(y_out, 0.5))) # (b, h)
451             y_out = self.fc2(F.dropout(y_out, 0.5)) # (b, num_classes)
452
453             if apply_softmax:
454                 y_out = F.softmax(y_out, dim=1)
455
456         return y_out
457
458         국가를 output
459
460
461
462
463
464
465
466
467
468
469
470
471
472

```

출처

<https://product.kyobobook.co.kr/detail/S000001810395>

파이토치로 배우는 자연어 처리 | 델립 라오 - 교보문고

파이토치로 배우는 자연어 처리 | 쉽고 빠르게 익히는 자연어 처리 입문 가이드북자연어 처리 (NLP)는 인공지능이 지닌 무한한 능력을 이용해 애플 시리, 아마존 알렉사, 구글 번역 등과 같은 제품

product.kyobobook.co.kr

파이토치로 배우는 자연어 처리

공감

구독하기

'practice_인공지능,머신러닝' 카테고리의 다른 글

CNN (0)

2024.10.16

Overfitting,Dropout,Weight Decay,Batch Normalization (0)

2024.10.16

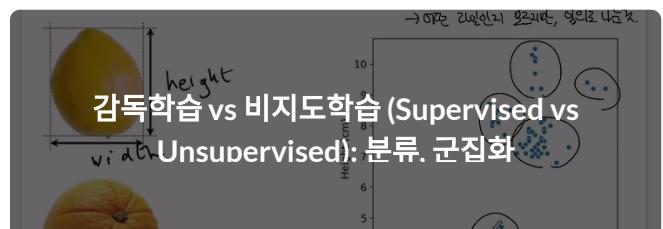
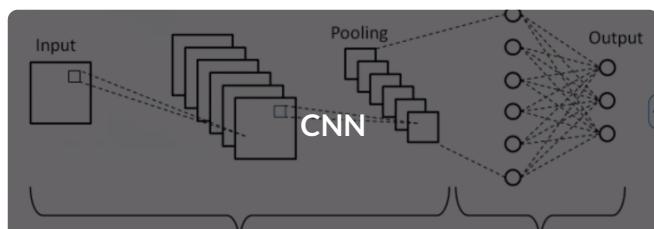
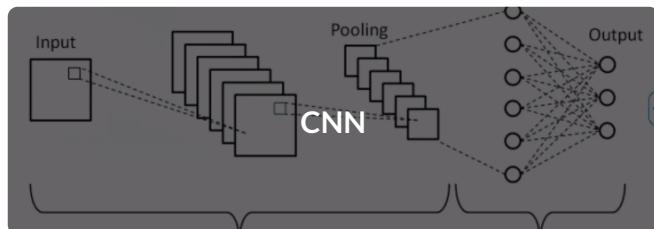
CNN (0)

2024.10.14

감독학습 vs 비지도학습 (Supervised vs Unsupervised): 분류, 군집화 (0)

2024.10.13

관련글

[관련글 더보기](#)

자연어(NLP)

네이쳐2024 님의 블로그입니다.

[구독하기 +](#)

댓글 0

이름

비밀번호

내용을 입력하세요.

[등록](#)



등록