인공지능개론 정리노트 #4
ICT융합공학부 202204010 공성택

6장

Q. 매개변수가 많을수록 오버피팅이 잘 일어나는 이유가 구체적으로 무엇인지 궁금하다.

A. 기본적으로 매개변수가 많다는 것은 신경망 모델이 복잡하다는 것을 의미한다. 복잡한 신경망은 훈련 데이터에 대해 그 특성을 지나치게 학습하기 때문에 불규칙적인 특성도 학습하게 된다. 따라서 훈련 데이터에만 잘 적용하고, 새로운 데이터들에는 잘 맞지 않는 현상이 일어날 수 있다. 그렇기 때문에 매개변수를 많이 설정할수록 여러 특징을 깊게 학습할 수 있지만 오버피팅으로 인해 새로운 데이터에 대한 성능이 떨어질 수 있고, 그렇다고 매개변수를 적게 설정할 경우에는 모델이 단순화되어 충분한 학습을 하지 못할 수 있다. 따라서 매개변수를 적절히 많이 사용하되, 드롭아웃 등의 방법을 통해 오버피팅을 방지할 수 있도록 하는 것이 효율적이고 적절한 설계 방법이라고 조사했다.

Q. 동일한 신경망 모델에서 동일한 데이터가 주어졌을 때, 최적의 하이퍼 파라미터는 항상 일정한 값으로 존재하는지 궁금하다.(실행에 따라서 변하지 않고 최적의 하이퍼 파라미터를 찾으면 다음 번의 테스트에서도 그 값이 최적의 하이퍼 파라미터인지)

A. 조사한 결과, 이론적으로 모든 조건이 동일하고 변동성이 없는 경우에는 최적의 하이퍼 파라미터가 일정한 값으로 존재할 수 있다. 그러나 실제 학습에서는 변동성이 있는 경우가 빈번하다. 예를 들면 확률적 학습 알고리즘을 사용하는 경우 매 테스트에서 무작위 업데이트를 하기 때문에 학습 결과가 다를 수 있다. 이외에도 가중치 초기화 과정 등에서 무작위 초기화가 되기도 하기 때문에 최적의 하이퍼 파라미터가 변동될 수 있다고 조사했다. 모든 조건과 데이터가 동일한 경우에는 이론적으로 하이퍼 파라미터가 일정한 값으로 존재할 수 있다는 것이 내가 찾은 결론이다.

7장

Q. CNN에서 풀링을 거치는 이유에 대해 더 자세하게 알고싶다.

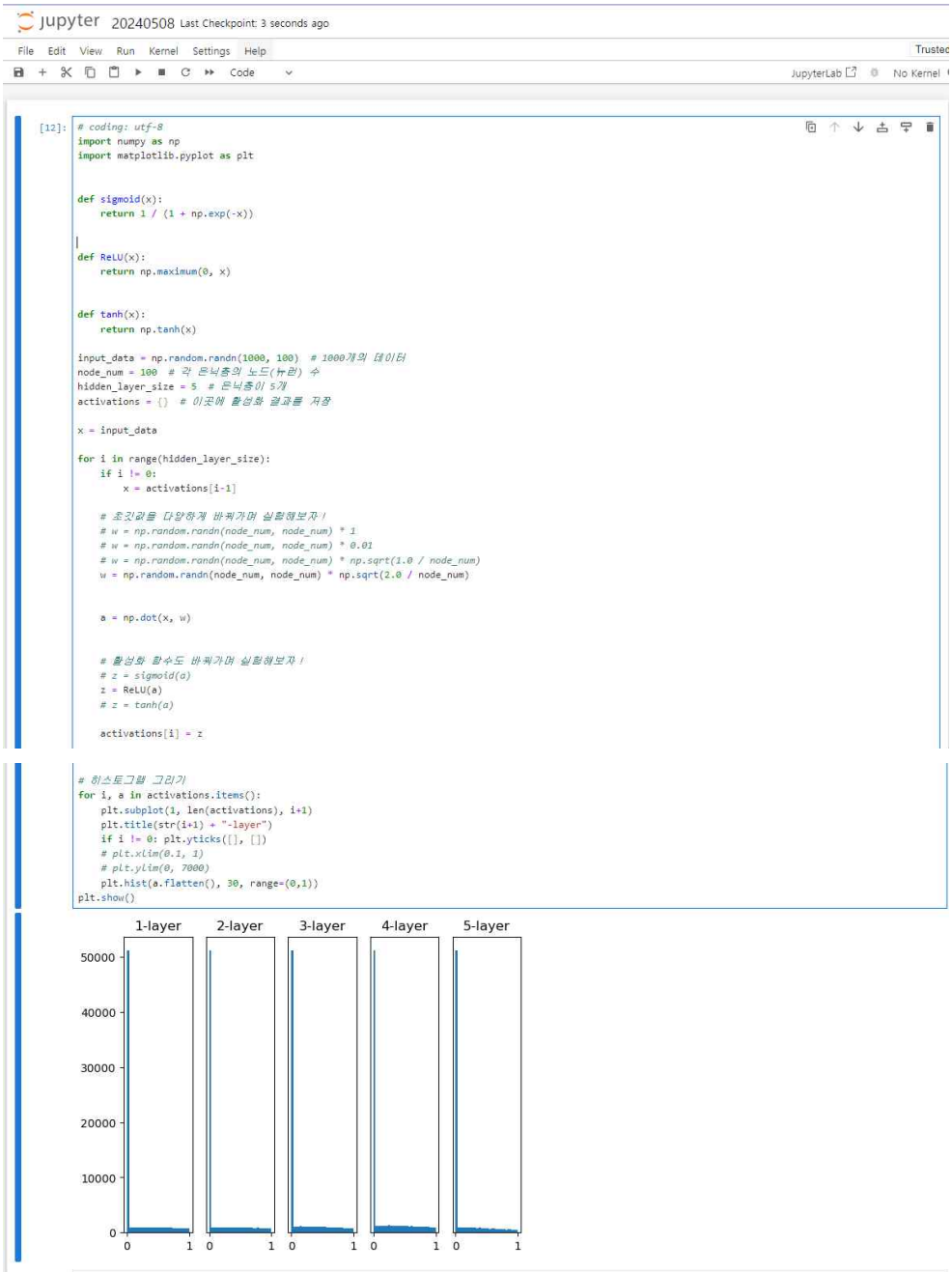
A. 풀링을 사용하면 데이터의 크기를 줄일 수 있다. 데이터를 줄임으로써 다양한 이점을 얻을 수 있는데, 가장 기본적으로 필요한 연산량이 줄어든다는 것이다. 그로 인해 메모리 사용량과 소요 시간 등이 줄어들 수 있다. 또, 특히 중요한 점은 과적합을 방지할 수 있다는 것이다. 수업 시간에 배운 최대 풀링을 사용하면 가장 큰 값만 추출한다. 따라서 신경망이 훈련 데이터의 작은 특징들을 학습하지 않게 되어 일반 데이터에 적용했을 때 과적합이 잘 발생하지 않도록 할 수 있다. 강의자료 설명만 봤을 때는 모델의 성능이 향상된다고 나와있는데 더 조사해보니 과적합 등의 이유에서 더 쉽게 이해할 수 있었다.

Q. 스트라이드를 크게 함과 동시에 패딩을 크게 하면 출력 크기는 어떻게 될 지 궁금하다. 또한 출력 크기 계산을 했을 때 정수로 나누어 떨어지지 않는 경우는 어떻게 될 지 궁금하다.

A. 결과는 각 요소의 크기 증가값에 대해서 달라질 수 있다. 수업 시간에 배운 공식을 이용해 계산할 수 있는데, 스트라이드를 크게 하는 것과 패딩을 크게 하는 것을 동시에 적용할 경우 두 요소의 영향이 동시에 작용한다. 예를 들면 패딩을 키워도 스트라이드의 영향 때문

에 값이 변하지 않는 경우가 있는데 이는 스트라이드의 영향이 더 커서 패딩이 그 영향을 상쇄하지 못한 경우이다. 따라서 출력 크기는 두 요소의 상대적인 크기에 따라 나타난다.
만약 출력 크기를 계산할 때 정수로 나누어 떨어지지 않는 경우가 나타날 수 있는데, 조사한 결과 이 때는 반올림이나 소수점 이하를 버리면서 정수로 만들 수 있다고 한다. 모델의 목적에 따라서 더 나은 방법을 선택한다고 조사했다.

6장_1) 수업시간 실습



6장_2) 하이퍼 파라미터 종류 생각해보기

6.5 적절한 하이퍼파라미터 값 찾기

- **하이퍼파라미터**
  - 각 층의 뉴런 수, 배치 크기, 매개변수 갱신 시의 학습률과 가중치 감소 등등
  - 신경망에서 다수 등장함

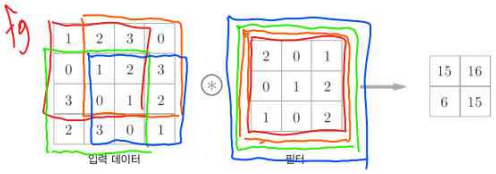
하이퍼 파라미터: 계층수, 층안 뉴런수, 학습률, 배치크기, 에포크수, 활성화함수, 옵티마이저, 드롭아웃 비율, 정규화 파라미터 등

- **하이퍼파라미터 값 설정**
  - 적절하게 설정하지 않으면 모델의 성능이 크게 떨어짐
  - 그 값을 결정하기까지는 일반적으로 많은 시행착오를 겪음
  - 하이퍼파라미터의 값을 최대한 효율적으로 탐색하는 방법 필요

7장_1) 합성곱 연산 직접 계산해보기

7.2.2 합성곱 연산

- **합성곱 연산** 정리노트
  - 이미지 처리에서 말하는 필터 연산에 해당함



  - 입력 데이터에 필터를 적용 (필터를 커널이라 칭함)
  - 입력은 (4, 4), 필터는 (3, 3), 출력은 (2, 2) 가 됨
  - CNN 에서는 필터의 매개변수가 가중치임

계산해보기

① 1×2 + 2×0 + 3×1 + 0×0 + 1×1 + 2×2 + 3×1 + 0×0 + 1×2
= 2 + 0 + 3 + 0 + 1 + 4 + 3 + 0 + 2
= 15

② 2×2 + 3×0 + 0×1 + 1×0 + 2×1 + 3×2 + 0×1 + 1×0 + 2×2
= 4 + 0 + 0 + 0 + 2 + 6 + 0 + 0 + 4
= 16

③ 0×2 + 1×0 + 2×1 + 3×0 + 0×1 + 1×2 + 2×1 + 3×0 + 0×2
= 0 + 0 + 2 + 0 + 0 + 2 + 2 + 0 + 0
= 6

∴ | 15 | 16 |
| 6 | 15 |

④ 1×2 + 2×0 + 3×1 + 0×0 + 1×1 + 2×2 + 3×1 + 0×0 + 1×2
= 2 + 0 + 3 + 0 + 1 + 4 + 3 + 0 + 2
= 15

7장_1) 예제1 출력 크기 수식으로 계산해보기

7.2.4 스트라이드(1)

- **스트라이드를 적용하여 출력 크기 계산**
  - (7, 7)인 입력 데이터에 스트라이드를 2로 설정하면 출력은 (3, 3) 이 됨
  - 스트라이드를 키우면 출력 크기는 작아지고, 패딩을 크게 하면 출력 크기가 커짐
  - 스트라이드, 패딩 관계를 이용한 출력 크기 계산을 수식화
    - 입력 크기를 (H, W), 필터 크기를 (FH, FW), 출력 크기를 (OH, OW), 패딩을 P, 스트라이드를 S 라 하면 출력 크기는 아래의 식으로 계산됨

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

  - 예제1) 입력 크기 = (4, 4), 패딩 = 1, 스트라이드 = 1, 필터 = (3, 3) 일때 출력 크기는?
  - 예제2) 입력 크기 = (7, 7), 패딩 = 0, 스트라이드 = 2, 필터 = (3, 3) 일때 출력 크기는?
  - 예제3) 입력 크기 = (28, 31), 패딩 = 2, 스트라이드 = 3, 필터 = (5, 5) 일때 출력 크기는?
  - 출력 크기가 정수가 아니면 반올림하는 등 에러를 내지 않도록 구현

7장_2) 05.29 수업 테스트 내용

jupyter 20240529 Last Checkpoint: 6 hours ago

File Edit View Run Kernel Settings Help

Code

JupyterLab Python 3 (ipykernel)

```
[5]: import numpy as np

     x = np.random.rand(10, 1, 28, 28)
     x.shape
```

```
[5]: (10, 1, 28, 28)
```

```
[6]: x[0].shape
```

```
[6]: (1, 28, 28)
```

```
[7]: x[0,0]
```

```
[7]: array([[6.19815108e-01, 1.35394771e-01, 1.18296229e-01, 1.12449707e-01,
            7.79045603e-01, 1.27447947e-02, 7.04321958e-02, 8.16593956e-01,
            3.56809175e-01, 2.88928039e-02, 6.39025976e-01, 6.14065456e-01,
            4.30303754e-01, 5.43632887e-01, 2.29778347e-01, 7.08496362e-01,
            3.90637074e-01, 1.17025404e-01, 9.97889773e-02, 8.35093521e-01,
            5.22623344e-01, 3.12356418e-01, 4.66641798e-01, 5.30080831e-02,
            4.91788691e-01, 2.53119492e-01, 5.40030229e-01, 1.78607829e-01],
           [6.98561092e-01, 6.19603324e-01, 4.41165653e-01, 9.63037359e-01,
            3.06331279e-02, 6.70463541e-01, 2.80826671e-01, 5.74926688e-01,
            6.75108413e-01, 9.97995695e-01, 6.83148285e-01, 2.56969306e-01,
            4.47187442e-01, 3.15787350e-01, 2.46210668e-01, 9.90380376e-01,
            8.31963700e-01, 5.09325426e-01, 5.62764022e-01, 7.40062004e-01,
            8.33044740e-01, 1.25495979e-01, 6.68991368e-01, 9.69341909e-01,
            7.24881535e-01, 9.39283135e-01, 5.05333103e-01, 8.90143601e-01],
           [3.61781436e-01, 4.48573827e-01, 7.19110036e-01, 7.37768610e-01,
            6.76078590e-01, 8.60575304e-01, 9.46893169e-01, 2.53939416e-01,
            3.99998084e-01, 6.91857590e-01, 5.36891186e-01, 5.54121692e-01,
```

```
[17]: import sys, os
      import numpy as np
      sys.path.append(os.pardir)
      from common.util import im2col

      x1 = np.random.rand(1, 3, 7, 7)
      col1 = im2col(x1, 5, 5, stride=1, pad=0)
      print(col1.shape)

      x2 = np.random.rand(10, 3, 7, 7)
      col2 = im2col(x2, 5, 5, stride=1, pad=0)
      print(col2.shape)
```

```
(9, 75)
(90, 75)
```

```
[19]: class Convolution:
          def __init__(self, W, b, stride=1, pad=0):
              self.W = W
              self.b = b
              self.stride = stride
              self.pad = pad

          def forward(self, x):
              FN, C, FH, FW = self.W.shape
              N, C, H, W = x.shape
              out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
              out_w = 1 + int((W + 2*self.pad - FW) / self.stride)

              col = im2col(x, FH, FW, self.stride, self.pad)
              col_W = self.W.reshape(FN, -1).T
              out = np.dot(col, col_W) + self.b

              out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

              return out
```

```
[20]: class Pooling:
          def __init__(self, pool_h, pool_w, stride=1, pad=0):
              self.pool_h = pool_h
              self.pool_w = pool_w
              self.stride = stride
              self.pad = pad

          def forward(self, x):
              N, C, H, W = x.shape
              out_h = int(1 + (H - self.pool_h) / self.stride)
              out_w = int(1 + (W - self.pool_w) / self.stride)

              col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
              col = col.reshape(-1, self.pool_h*self.pool_w)

              out = np.max(col, axis=1)

              out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

              return out
```