모델 개선

최 석 재

lingua@naver.com

Weight Regularization

간단한 모델과 복잡한 모델

- 모델이 필요 이상으로 많은 노드를 가지고 있으면 모델이 복잡해진다
- 모델이 복잡해지면 과대적합이 빨리 이루어진다
- 따라서 모델의 일반화 성능이 떨어진다

- 그러므로 모델의 복잡도에 제한을 두어 모델이 덜 복잡해지게 만든다
- 그 방법으로는 손실 계산에 비용(penalty)을 추가하여 가중치를 낮춘다
- 특정 노드의 가중치를 0에 가깝게 만들면 실제로는 노드가 없어지는 효과를 갖는다
- 특이한 데이터 포인트는 그것을 설명하기 위해 특정 노드에 높은 가중치를 두게 된다
- 이러한 것들이 규제의 주요 대상이며, 비용을 추가하여 가중치를 많이 줄인다

- L1 규제: 가중치의 절댓값에 비례하는 비용을 추가한다
- L2 규제: 가중치의 제곱에 비례하는 비용을 추가한다

$$\sum_{i=1}^{n}(y_i-\hat{y}_i)^2+\alpha\sum_{n=1}^{N}|\theta_n|$$

$$\sum_{i=1}^{n}(y_i-\hat{y}_i)^2+\alpha\sum_{n=1}^{N}\theta_n^2$$

※ θ : 가중치

규제 설정

- 다음과 같이 모델을 설정할 때 규제를 설정할 수 있다
- 규제의 기본값은 0.01 이다

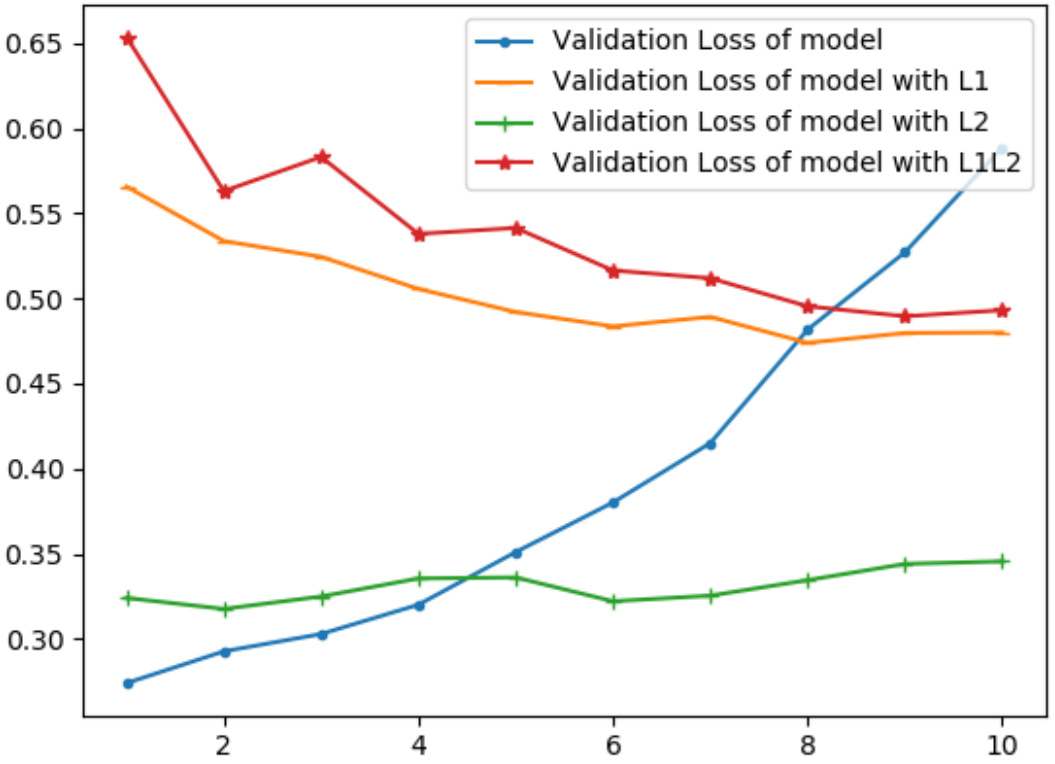
```
# 기본 모델
model = Sequential()
model.add(Dense(16, activation='relu', input_shape=(max_words,)))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


# l1 규제 모델
model_l1 = Sequential()
model_l1.add(Dense(16, kernel_regularizer=regularizers.l1(0.001), activation='relu', input_shape=(max_words,)))
model_l1.add(Dense(8, kernel_regularizer=regularizers.l1(0.001), activation='relu'))
model_l1.add(Dense(1, activation='sigmoid'))


# l2 규제 모델
model_l2 = Sequential()
model_l2.add(Dense(16, kernel_regularizer=regularizers.l2(0.001), activation='relu', input_shape=(max_words,)))
model_l2.add(Dense(8, kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model_l2.add(Dense(1, activation='sigmoid'))
```

```
# l1 l2 규제 병행 모델
model_l1l2 = Sequential()
model_l1l2.add(Dense(16, kernel_regularizer=regularizers.l1_l2(l1=0.001, l2=0.001), activation='relu', input_shape=(max_words,)))
model_l1l2.add(Dense(8, kernel_regularizer=regularizers.l1_l2(l1=0.001, l2=0.001), activation='relu'))
model_l1l2.add(Dense(1, activation='sigmoid'))
```

결과 확인



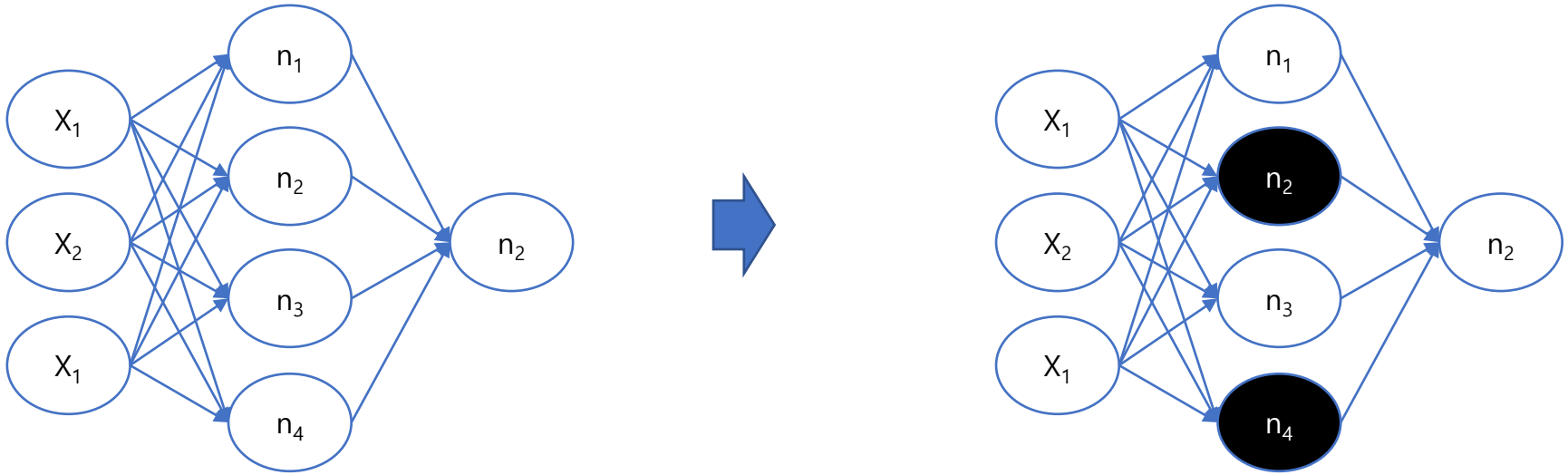
- IMDB 이진분류 데이터를 이용하여 비교하였다

- 검증데이터에 대하여 기본 모델의 손실값은 빠르게 올라간다
- L1 규제를 한 모델은 처음부터 손실값이 높은 편이다
- L1 규제와 L2 규제를 함께 쓰는 방식을 ElasticNet 이라고도 한다

- L2 규제를 한 모델의 손실값이 가장 천천히 올라간다

Dropout

Dropout 원리



- Dropout은 노드의 일부분을 사용하지 않는 기법이다
- 그 결과 과대적합을 막을 수 있다

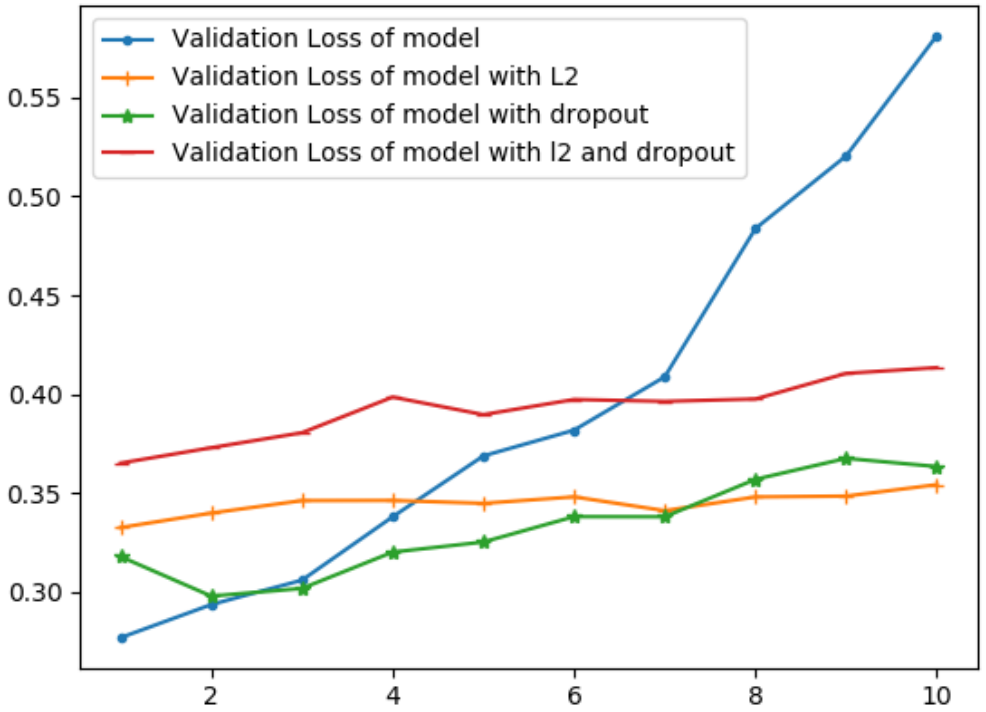
- 각 샘플에 대해 뉴런의 일부를 무작위하게 제거하면 뉴런의 불필요한 협업을 방지하게 된다
- 즉, 중요하지 않은 우연한 패턴은 깨뜨리게 된다

Dropout 설정

- 단순히 층의 출력 바로 뒤에 Dropout 층을 추가하면 된다

```
model_drop = Sequential()
model_drop.add(Dense(16, activation='relu',input_shape=(max_words,)))
model_drop.add(Dropout(0.2))
model_drop.add(Dense(8, activation='relu'))
model_drop.add(Dropout(0.5))
model_drop.add(Dense(1, activation='sigmoid'))
```

결과 확인



- Dropout을 적용한 모델이 과대적합이 늦게 이루어진다

- L2 규제를 같이 사용하면 손실값이 다소 크게 나타나고,
- L2 규제만 사용하는 것은 다소 비슷하다

파라미터 조정

규제 옵션

- 규제 옵션으로는 앞에서 살펴본 바와 같이 다음과 같은 것이 있다

- L1, L2 규제
- Dropout

함수 선택

- 모델의 마지막 층에 적용될 활성화 함수와 손실 함수는 문제의 유형에 따라 달라진다
- 일반적으로 선택되는 함수의 종류는 다음과 같다

문제 유형	마지막 층의 활성화 함수	손실 함수
이진 분류	sigmoid	binary_crossentropy
다중 분류	softmax	categorical_crossentropy
임의값 회귀 분석	None	mse
0~1 사이값에 대한 회귀	sigmoid	mse 또는 binary_crossentropy

Optimizer & Learning Rate

- 옵티마이저는 rmsprop/adam, 학습률은 기본값을 사용하는 것이 무난하다

Optimizer	Learning Rate
RMSprop	기본값
Adam	기본값

- https://keras.io/optimizers/
- SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam

- RMSProp은 수정량을 처음에는 크게 하다가, 점차 줄여가며 최적의 가중치를 찾는 기법이다
- Adam은 RMSProp보다 더 부드럽게 수정량을 줄여나간다

Node

- 그 외에 다음의 옵션을 통해 모델을 더 키워서 최고 성능의 모델을 찾는다

- 층을 추가해본다
- 각 층의 노드를 키워본다
- 더 많은 에포크 동안 훈련한다

연결된 순환신경망

순환 드롭아웃 *recurrent dropout*

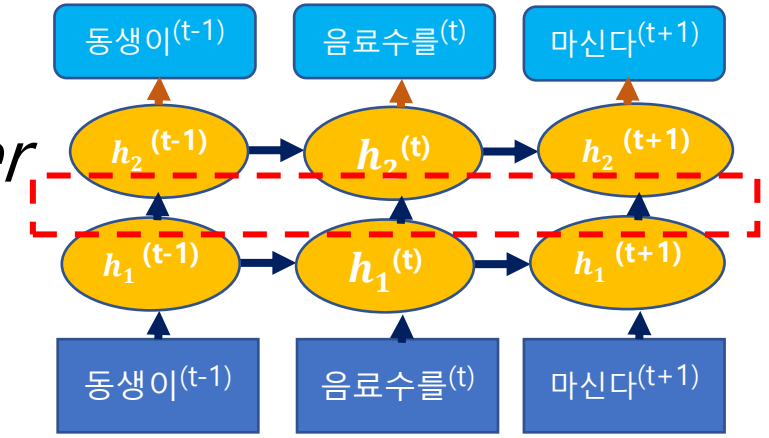
- 장점: 순환층에서 과대적합을 방지한다

- 순환층에 드롭아웃을 적용한다
- 일반적으로 층의 입력과 동일한 드롭아웃 마스크를 적용한다
- 드롭아웃 특성상 과대적합은 잘 안 일어나지만, 성능 개선은 크지 않을 때가 많다

- dropout : 층의 입력에 대한 드롭아웃 비율
- recurrent_dropout : 순환 상태(타임 스텝)의 드롭아웃 비율

- `model.add(GRU(16, dropout=0.2, recurrent_dropout=0.2))`

스태킹 순환층 *stacking recurrent layer*

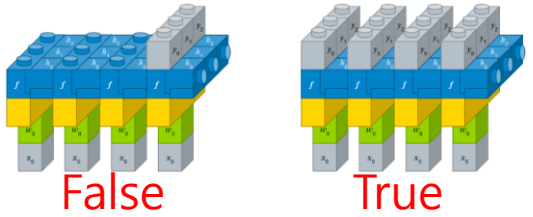


- 장점: 네트워크의 표현능력을 증가시킨다

- 순환 드롭아웃에서 성능을 올리기 위하여 네트워크의 용량을 올리는 기법
- 순환층을 쌓기 위해서는 전체 시퀀스를 출력해야 한다
- 성능을 실제 올리려면 층을 많이 쌓아야 한다. 단, 계산 비용이 크게 든다

- return_sequences=True : 현재 층의 모든 타임스텝 결과를 출력하여, 다음 층이 입력으로 사용할 수 있게 한다

```
model.add(GRU(16, dropout=0.1, recurrent_dropout=0.5, return_sequences=True))
model.add(GRU(8, dropout=0.1, recurrent_dropout=0.5))
```



양방향 순환층 *bidirectional recurrent layer*

- 장점: 기억을 좀더 오래 유지시킨다

- 한쪽 방향으로 진행하는 RNN과 다른 방향으로 진행하는 RNN을 결합한다
- 그러면 새로운 표현이 발견되어, 새로운 패턴이 학습될 수 있다
- 양방향 순환층도 스태킹할 수 있다

- `model.add(Bidirectional(LSTM(16))`

```
model = Sequential()
model.add(Embedding(max_words, embedding_dim, maxlen))
model.add(Bidirectional(LSTM(32, return_sequences=True)))
model.add(Bidirectional(LSTM(16)))
model.add(Dense(8, activation='softmax'))
```