# 6주차

| 태그 | 완료 |
| --- | --- |

## Train/dev/test

- applied ML is a highly **iterative process**
  - we have to set hyperparameters like #layers, #hidden units, learning rate, activation function in order to train nn
  - idea→code→experiment→idea→code→experiment→...
  - NLP, CV, speech, structural data 등등 다양한 분야가 있지만 각자 분야에서의 직관이 다른 분야에는 적용되지 않는다.

💡**이 과정에서 훈련/개발/테스트 데이터셋을 잘 설정하면** iterative process를 효율적으로 만들 수 있다.

**Data 비율**

| **train set** | **dev set** | **test set** |
| --- | --- | --- |

- ML에서 데이터셋은 3개의 종류로 나뉜다→ train/dev/test
- train set는 모델을 훈련하고 dev set로 그 모델의 성능을 검증하고 개선한다. 최종적으로 완성된 모델을 test set을 이용해 그 성능을 평가한다.
- 이전에는 60%/20%/20%의 비율로 데이터를 나누는 것이 가장 좋다고 여겨졌지만, 100만 데이터를 다루는 요즘에는 dev set와 test set의 비율을 줄이는 것이 좋다고 한다. 따라서 dev set는 여러 개의 알고리즘 모델 중 어떤 것이 좋은지 평가할 수 있을 정도만 있으면 된다. test set 역시 많이 필요 없다. 결론은, 100만 개의 샘플이 있다면 dev set와 test set에 각각 1만개 정도의 샘플만 있어도 된다는 거다. 비율은 98%/1%/1%이다.
- Mismatched train/test distribution

  -make sure dev set and test set come from **same distribution** with train

- Not having a test set might be okay(Only dev set)
  - test에 해당하는 데이터를 train에서 학습시키고 dev로 성능 평가만 한다.
  - 흔히 train/test라고 하지만, 사실 train/dev가 올바른 표기다.

## 편향/분산

- In case of cat classification(cat=1/not cat=0)
- 가정: 인간 수준의 성능이 기본이 되어야 한다. 인간은 개와 고양이를 0% 오차로 구분할 수 있다. 더 일반적으로 말하면, 베이지안 최적 오차가 0%라는 설정이 깔려 있다. (인간이 잘 구별한다==베이즈 오차가 크지 않다)

| | |
|---|---|
| 높은 편향(high bias) | 과소적합(underfitting) |
| 알맞음(just right) | - |
| 높은 분산(high variance) | 과대적합(overfitting) |

| | 높은 분산<br>(과대적합) | 높은 편향<br>(과소적합) | 높은 편향 &<br>높은 분산 | 낮은 편향 &<br>낮은 분산 |
|---|---|---|---|---|
| 훈련 세트 | 1 % | 15 % | 15 % | 0.5 % |
| 개발 세트 | 11 % | 11 % | 30 % | 1 % |

- 훈련 세트는 오차가 크지 않지만 개발 세트에서 오차가 클 경우→overfitting(높은 분산)...variance(분산)이므로 precision(정밀도)의 문제:높은 분산==낮은 정밀도
- 훈련 세트의 오차가 클 경우→underfitting(높은 편향)...biased(편향)이므로 accuracy(정확도)의 문제: 높은 편향==낮은 정확도
- 훈련 세트의 오차도 크고 개발 세트의 오차도 큰 경우→높은 분산&높은 편향→최악
- 훈련 세트와 개발 세트의 오차가 크지 않은 경우→낮은 분산&낮은 편→적절

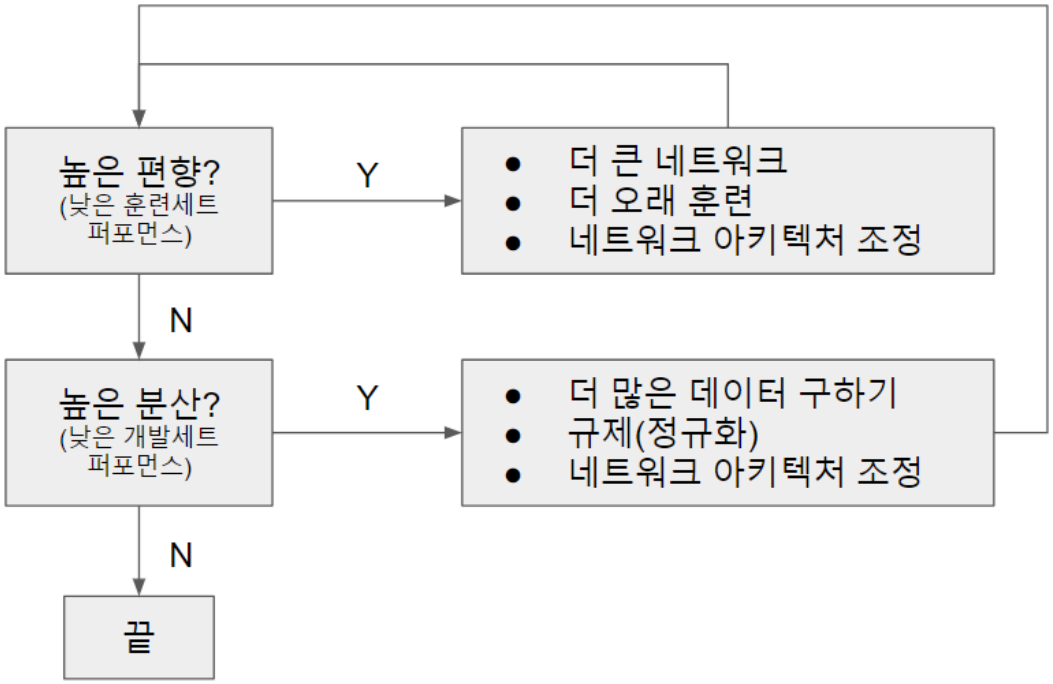
## 머신러닝을 위한 기본 레시피

- 데이터가 높은 편향을 갖는지 확인하려면 training data performance를 봐야 한다.

➡️ 해결책: bigger network, train longer, try with another ML architecture

- 데이터가 높은 분산을 갖는지 확인하려면 dev data performace를 봐야 한다.

➡️ 해결책: more data, regularization, try with another ML architecture




💡이전에는 편향을 줄이면 분산이 커지고 분산을 줄이면 편향이 커지는 트레이드오프(trade-off)가 있었다. 그러나 딥러닝 빅테이터 시대에는 bigger network를 사용해도 분산이 커지지 않고, data를 더 모아도 편향이 커지지 않는다. 서로 서로 영향을 안 미칠 수 있는 뜻이다.

## 정규화

- 높은 분산으로 overfitting의 문제가 있다면, 가장 먼저 고려해야 할 해결책은 정규화(regularization)이다. (더 많은 data를 모을 수도 있지만, 이는 시간과 비용이 들기 때문에 함수의 복잡도를 줄이는 regularization을 선호한다)
- logistic regression
  - objective: $min_{w,b}J(w,b)$
  - L2 regularization(weigth decay): $||w||_2^2 = \sum_{j=1}^{n} w_j^2 = w^T w$
  - L1 regularization: $||w|| = \sum_{j=1}^{n} |w_j|$

        - L1 regularization은 w가 sparse(사라진다)된다고 해서 inherent feature selection을 한다. 즉 일부 특징들은 반영되지 않는다. 통상적으로 L2가 L1보다 많이 사용된다.
        - 왜 w에 대해서만 정규화할까? b는?

          다차원의 경우 w는 높은 차원을 갖지만 b는 하나의 숫자이기 때문에 실질적으로 영향을 거의 미치지 않는다. 즉, 대부분의 파라미터는 w이기 때문에 w만 regularization term에 반영한다.
        - $\lambda$(regularization strength)는 정규화 하이퍼파라미터이다.
    - cost function with L2 regularization: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}||w||_2^2$
    - Frobenius norm: $||w^{[l]}||_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} w_{ij}^2$
        - 행렬의 모든 원소를 제곱해서 더한 것이다.
        - Frobenius norm은 행렬을 1차원 벡터로 만들어 L2 norm을 구하는 것이다.
    - L2 regularization이 weight decay라고 불리는 이유는?
        - $w^{[l]} = (1 - \frac{\alpha\lambda}{m})w^{[l]} - \alpha$*(역전파에서 온 값들)
        - 위의 식에서 w에 1보다 작은 값인 $(1 - \frac{\alpha\lambda}{m})$이 곱해지기 때문이다.

      ➡️ **큰 가중치에 대해 penalty($\lambda$)를 부과하여 가중치의 크기를 줄이는 개념**이다. 따라서 이런 **penalty term을 줄이기 위해 가중치 행렬 W을 줄이게 되는 것이다 →weigth decay**

## 왜 정규화는 overfitting을 줄일 수 있을까?

1. **정규화에서 $\lambda$를 크게 하면 가중치 행렬 W를 0에 상당히 가깝게 할 수 있다.**

   $\lambda$가 커지면 regularization term이 전체 식에 끼치는 영향이 커진다. 네트워크에서는 이를 방지하기 위해 가중치 행렬 W를 0에 가깝게 한다. 이런 방식으로 많은 은닉 유닛의 영향을 줄여 나가 로지스틱 회귀 신경망처럼 만들면, 신경망은 깊은 층을 포함하면서도 간단한 신경망이 될 것이다. 간단한 네트워크는 overfitting 문제를 완화한다.
2. 만약 은닉층의 활성화 함수($g$)로 tanh를 사용하는 경우, tanh의 중간 부분은 선형에 가깝다. 이때 $\lambda$를 크게 하면 가중치 행렬 W는 0에 가까워지고, $z^{[l]} = W^{[l]}a^{[n-1]} + b[l]$도 작아진다. 그 결과 $z^{[l]}$는 활성화 함수 $g$(tanh)의 중간 부분을 통과하므로 선형에 가까울 것이다. **선형 활성화 함수를 사용하면 전체 네트워크도 선형이 되므로 복잡한 함수를 표현하는 것이 어려워진다.** 따라서 overfitting 될 때의 구불구불하고 복잡한 boundary가 그려지지 않기 때문에, overfitting을 방지할 수 있다.

💡 **공통 아이디어**

λ를 크게 해 가중치 행렬 W의 크기를 줄인다. 줄이게 되면...

1. 전체 네트워크는 깊으면서도 간단한 신경망이 된다.
2. tanh를 활성화 함수로 사용하는 경우 z는 tanh 중앙의 선형 부분으로 계산되기 때문에 전체 네트워크를 선형적으로 표현한다.

➡️ 간단하고 선형적인 신경망은 복잡한 함수를 그리지 못하기 때문에 overfitting을 방지할 수 있다.

## 드롭아웃 정규화

- 드롭아웃은 L2 정규화 외에 강력한 정규화 기법이다.
- 드롭아웃은 각각의 층에 대해 노드를 삭제하는 확률을 설정하는 것으로, 신경망의 하이퍼 파라미터이다. 삭제할 노드를 랜덤으로 선정 후 삭제된 노드와 연결된 입출력 링크도 모두 삭제한다.
- 이 과정을 여러 번 반복하면 신경망은 간단해진다. 그냥 봤을 때는 무작위로 노드를 삭제하는 엉뚱한 방법처럼 보이겠지만, 실제로 잘 작동한다.
- 드롭아웃 기법
  - 역 드롭아웃(Inverted Dropout): 가장 일반적인 드롭아웃 기법으로, 노드를 삭제 후 **얻은 활성화 값에 keep_prob(노드를 삭제하지 않을 확률)로 나눠주는 것**이다. 이는 **기존에 노드를 삭제하지 않았을 때 활성화 값의 기댓값과 값을 맞춰주기 위해서이다.**

```
# 3 layer
# a는 활성화 값

keep_prob=0.8 # 노드가 삭제되지 않을 확률

d3=np.random.rand(a3.shape[0], a3.shape[1])<keep_prob #boo
a3*=d3 # 삭제되지 않는(1) 값만 남게 된다
a3/=keep_prob
```

- 역전파 단계에서도 드롭아웃된 노드는 사용하지 않는다.
- 테스트 단계에서는 드롭아웃을 사용하지 않는다.

## 드롭아웃의 이해

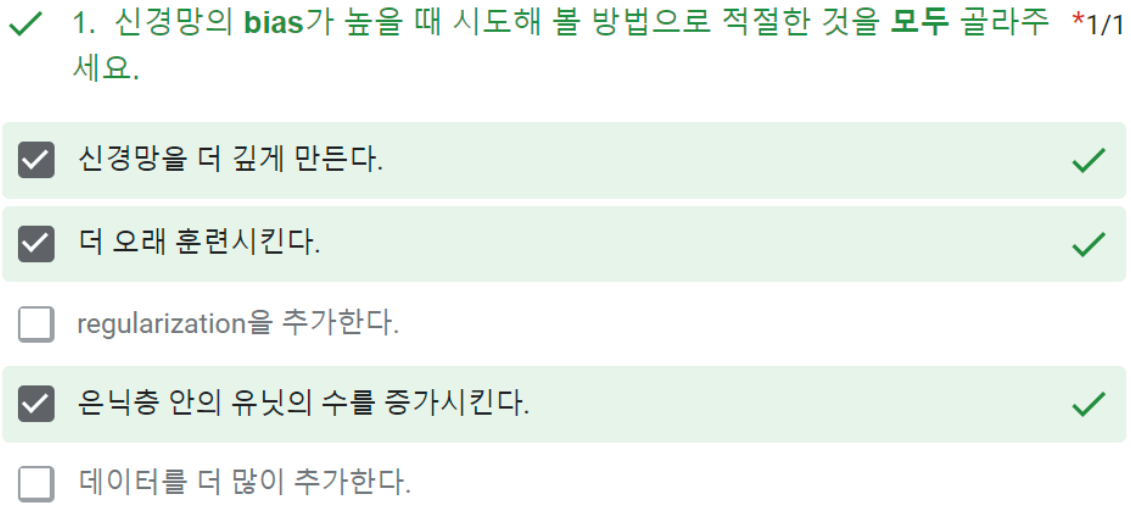
- 드롭아웃을 통해 노드들이 삭제되면 간단한 신경망이 된다. 따라서 모든 반복마다 기존보다 더 작은 신경망에서 작업하게 되고 이것이 정규화 효과를 낸다. 예를 들어 4개 노드-1개 노드가 있다고 한다면, 4개의 노드는 드롭아웃을 통해 무작위로 삭제된다. 따라서 **1개 노드는 4개 노드 중 어느 하나의 노드의 특성에도 의존하지 못하게 된다.** 즉 특정 입력에 큰 가중치를 부여하는 것이 꺼려지는 상황이 되고, 4개의 **입력 각각에 가중치를 고르게 분산시키게 된다.** 이는 **가중치 norm의 제곱값(L2 norm)이 줄어드는 효과를 내며 결국 overfitting을 완화**한다. 정리하자면 드롭아웃은 L2 regularization과 비슷한 효과를 낸다.
- 노드의 개수가 많은 층에는 keep_prob를 작게 해 삭제 효과를 더 내고, 노드의 개수가 적은 층에는 keep_prob를 크게 해 삭제 안 될 효과를 더 낼 수 있다. overfitting의 우려가 없는 층(입력층, 출력층)은 keep_prob=1로 설정해 드롭아웃을 적용하지 않을 수도 있다.
- CV 분야에서 최초로 드롭아웃이 성공한 경우가 많다. 그 이유는 CV에는 데이터가 충분하지 않기 때문에 overfitting이 많이 발생해 드롭아웃을 많이 사용하기 때문이다.
- 드롭아웃의 단점은 비용함수(J(w,b))가 잘 정의되지 않는다는 것이다. 모든 반복마다 랜덤으로 한 뭉치의 노드들을 삭제하게 되면 비용함수가 경사하강법을 통해 단조감소하는지 확인하기 어려워지기 때문이다. 이럴 땐 keep_prob=1로 설정해서 드롭아웃을 멈추고 J가 단조감소하는지(J가 잘 정의되는지) 확인한 후 드롭아웃을 사용해야 한다.

## 다른 정규화 방법들

- data augmentation: 이미지를 수평 방향으로 뒤집어 훈련 세트를 2배로 늘릴 수 있다. 혹은 이미지를 무작위로 확대/회전/왜곡시킬 수도 있다. 비록 겹치는 이미지들이 많아져 좋지 않고 유의미한 추가적 정보를 주지도 않지만, 큰 비용을 들이지 않고 데이터셋을 늘려 overfitting을 해결할 수 있다. 이미지뿐 아니라 digit의 경우도 마찬가지다.
- early stopping: 비용함수는 단조감소 형태로 그려진다. 조기 종료 기법에서는 train set error와 dev set error를 모두 그린다. dev set error가 증가하는 시점은 overfitting이 시작되는 점이므로 그 iteration을 찾아 중간에 반복을 멈춘다. 이때, 초기에 가중치 행렬 W은 0에 가까운 상태고 반복할수록 그 값이 커진다.  반복을 멈출 때 W는 중간 크기의 값을 갖는 상태다. 따라서 끝까지 반복을 진행했을 때보다 작은 W을 가중치 norm으로 사용하고 dev set error가 가장 작은 지점(모델이 가장 잘 작동하는 지점)에서 반복을 멈춤으로써 정규화 효과를 낼 수 있다.
    - 단점: 비용함수를 줄이는 동시에 overfitting을 막아야 한다. 그러나 만약 조기 종료를 한다면 비용함수를 최적화하지 못하게 된다. 원래 비용함수를 줄이는 것과 정규화는 각각 다른 방법으로 접근해야 하지만 조기 종료는 이 두 개의 문제 해결을 섞어

버리기 때문이다. 이 문제를 해결하기 위해 여러 개의 도구보다 하나의 도구를 사용하게 되지만, 이것은 문제를 복잡하게 한다.

## Quiz




- 헷갈린 것: 신경망의 bias가 높다는 것은 신경망이 낮은 정확도를 갖고 있음을 의미한다. 즉 underfitting 되었다는 뜻이다. 이를 해결하는 방법은 신경망을 더 깊게 만들고, 더 오래 훈련시키고, 모델의 표현력을 높이기 위해 은닉층의 유닛수를 증가시키는 것이 있다.