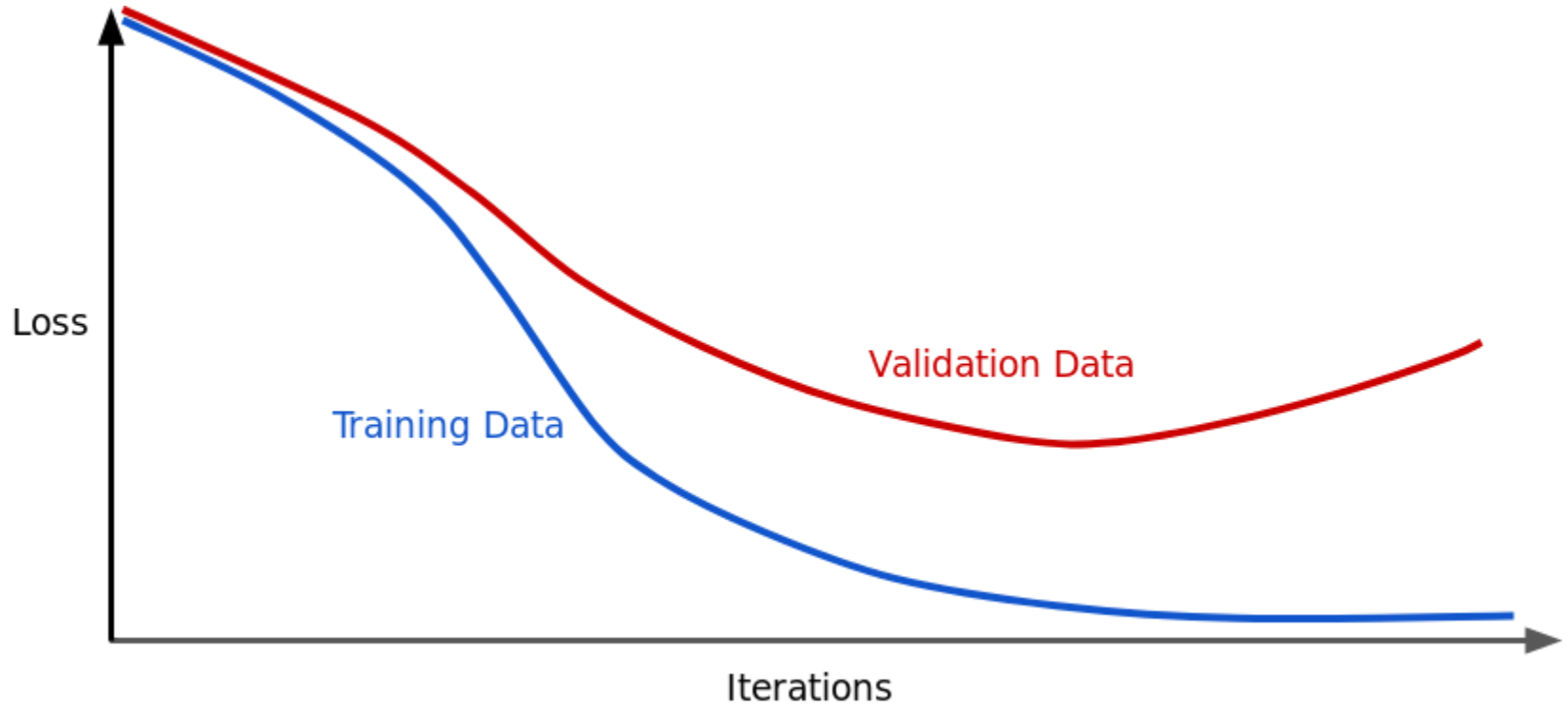
단순성을 위한 정규화: L_2 정규화

출처: https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/l2-regularization?hl=ko



그림은 Training Loss은 점차 감소하지만 Validation Loss은 결국 증가하는 모델을 보여줍니다.

즉, 이 일반화 곡선은 **모델이 training set의 데이터에 대해 overfitting하다는 것을 보여줍니다.**

다시 말해 다음은 단순히 Loss을 최소화하는 것만을 목표로 삼습니다 (empirical risk minimization).

minimize (Loss (데이터|모델))

이제 **structural risk minimization**를 통해 다음과 같이 Loss과 complexity를 함께 최소화해 보겠습니다.

minimize (Loss (데이터|모델) + complexity (모델))

이제 우리의 학습 최적화 알고리즘은 모델이 데이터에 얼마나 적합한지 측정하는 **Loss term**과 모델 복잡도를 측정하는 **regularization term**의 함수가 됩니다.

일반적인 (그리고 어느 정도 서로 관련이 있는) 2가지 방법으로 모델 복잡도를 다루게 됩니다.

- 모델의 **모든 특성의 *가중치*에 대한 함수**로서의 **모델 복잡도**
- 0이 아닌 가중치를 사용하는 *특성의 총 개수*에 대한 함수로서의 모델 복잡도 (L_1 정규화에서 이 접근 방식을 다룹니다.)

모델 복잡도가 가중치에 대한 함수인 경우, 높은 절대값을 사용하는 feature 가중치는 낮은 절대값을 사용하는 feature 가중치보다 더 복잡합니다.

모든 특성 가중치를 제곱한 값의 합계로서 정규화 항을 정의하는 ***L_2* regularization** 공식을 사용하여 복잡도를 수치화할 수 있습니다.

$$L_2 \text{ regularization term} = ||\boldsymbol{w}||_2^2 = w_1^2 + w_2^2 + \ldots + w_n^2$$

이 공식에서 0에 가까운 가중치는 model complexity에 거의 영향을 미치지 않는 반면, outlier 가중치는 큰 영향을 미칠 수 있습니다.

예를 들어 다음과 같은 가중치를 갖는 linear 모델이 있습니다.

{w1=0.2,w2=0.5,w3=5,w4=1,w5=0.25,w6=0.75}

위 모델의 L_2 regularization term은 다음과 같이 26.915입니다.

$$
\begin{aligned}
& w_1^2 + w_2^2 + \boldsymbol{w_3^2} + w_4^2 + w_5^2 + w_6^2 \\
&= 0.2^2 + 0.5^2 + \mathbf{5^2} + 1^2 + 0.25^2 + 0.75^2 \\
&= 0.04 + 0.25 + \mathbf{25} + 1 + 0.0625 + 0.5625 \\
&= 26.915
\end{aligned}
$$

하지만 제곱한 값이 25인 위의 굵은 글씨체로 된 w3는 거의 모든 복잡도에 기여합니다.

단순성을 위한 정규화: 람다

모델 개발자는 **람다**라는 스칼라(**regularization rate**이라고도 함)를 정규화 항의 값에 곱하여 정규화 항의 전반적인 영향을 조정합니다. 즉, 모델 개발자는 다음을 수행하는 것을 목표로 합니다.

$$\text{minimize}(\text{Loss}(\text{Data}|\text{Model}) + \lambda\, \text{complexity}(\text{Model}))$$

L_2 regularization를 수행하면 모델에 다음과 같은 효과를 줄 수 있습니다.

- 가중치 값을 0으로 유도(정확히 0은 아니고 가까운 값)
- 정규 (종 모양 또는 가우시안) 분포를 사용하여 가중치 평균을 0으로 유도

lambda 값을 높이면 regularization 효과가 강화됩니다. 예를 들어 높은 lambda 값에 대한 가중치 히스토그램은 그림 2처럼 보일 수 있습니다.

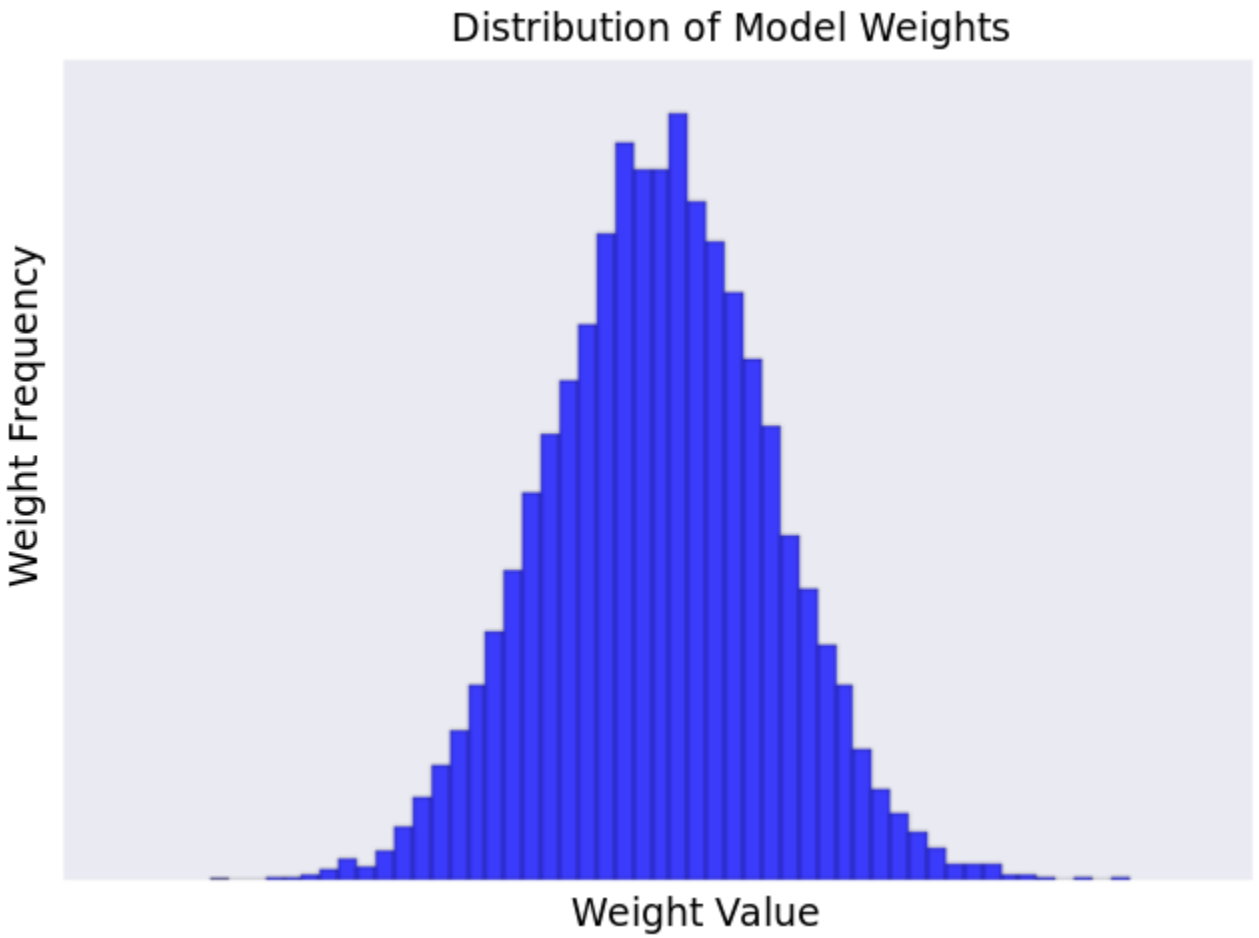


그림 2. 가중치 히스토그램

lambda 값을 낮추면 그림 3과 같이 더 평평한 히스토그램이 산출되는 경향이 있습니다.

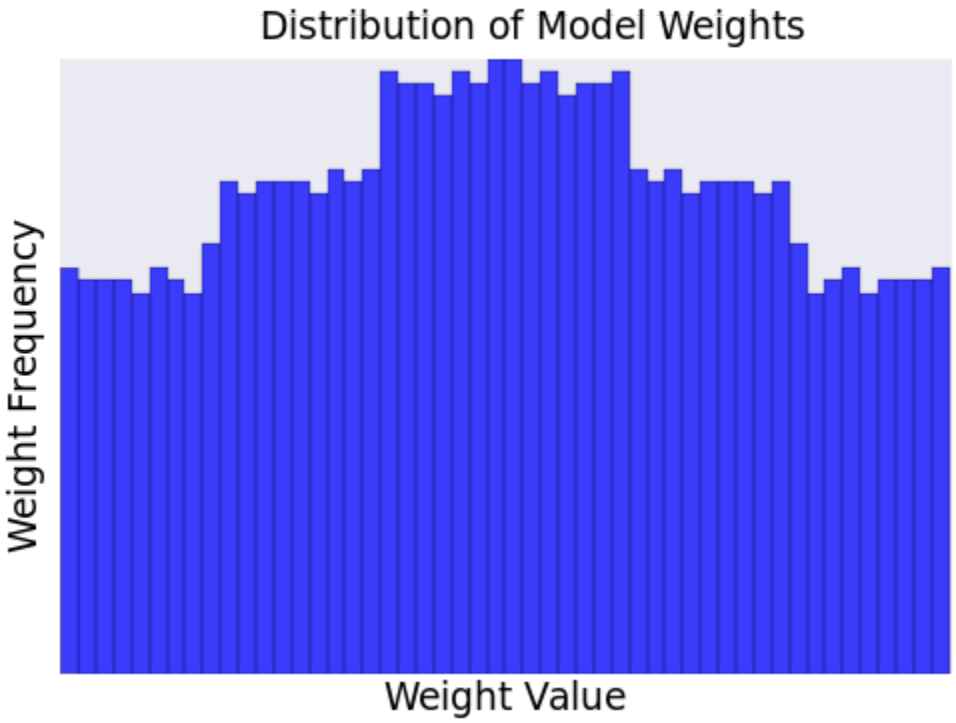


그림 3. 더 낮은 lambda 값으로 생성된 가중치 히스토그램

lambda 값을 선택할 때 세워야 할 목표는 simplicity과 training-data 적합성 사이에 적절한 균형을 맞추는 것입니다.

- lambda 값이 너무 높으면 모델은 단순해지지만 데이터가 *underfitting*해질 위험이 있습니다. 그렇게 되면 모델은 유용한 예측을 수행할 만큼 학습 데이터에 대해 충분히 학습하지 못할 수 있습니다.

- lambda 값이 너무 낮으면 모델은 더 복잡해지고 데이터가 *overfitting*해질 위험이 있습니다. 모델이 학습 데이터의 특수성을 너무 많이 학습하게 되고 새로운 데이터로 일반화하지 못하게 됩니다.

참고: 람다를 0으로 설정하면 정규화가 완전히 제거됩니다. 이 경우 학습이 손실을 최소화하는 데에만 초점을 맞추게 되어 가장 높은 수준의 과적합 위험을 낳습니다.

lambda의 이상적인 값은 이전에는 볼 수 없었던 새로운 데이터로 잘 일반화하는 모델을 생성합니다. 불행히도 이상적인 람다 값은 데이터에 따라 달라 지므로 약간의 tunning이 필요합니다.

L_2 정규화와 학습률

learning rate과 lambda는 밀접하게 연결되어 있습니다. 강력한 L_2 regularization 값은 feature 가중치를 0에 가깝게 유도하는 경향이 있습니다. 낮은 learning rates(early stopping 포함)도 종종 같은 효과를 가져오는데 이는 0과의 step 차이가 그다지 크지 않기 때문입니다. 결과적으로 learning rate과 lambda를 동시에 변경하면 혼동스러운 효과를 낳을 수 있습니다.

early stopping이란 모델이 완전히 수렴되기 전에 학습을 끝내는 것을 뜻합니다. 실제로 학습이 온라인 (연속적) 방식일 경우 일정 부분 암묵적으로 학습을 조기에 중단하는 경우가 많습니다. 즉, 일부 새로운 trends에는 아직 수렴을 위한 데이터가 충분하지 않습니다.

이미 언급했듯이 regularization parameters 변경으로 인한 효과는 learning rate 또는 반복 횟수의 변경으로 인한 효과와의 혼동을 일으킬 수 있습니다. **한 가지 유용한 방법(고정된 데이터 배치를 가지고 학습하는 경우)은 early stopping의 영향이 발생하지 않도록 반복 횟수를 충분히 높이는 것입니다.**

희소성을 위한 정규화: L_1 Regularization

희소 벡터는 종종 많은 차원을 포함합니다. 특성 교차(feature cross)를 생성하면 더 많은 차원이 발생합니다. 이러한 고차원 특성 벡터가 주어지면 모델 크기가 커질 수 있으며 엄청난 양의 RAM이 필요합니다.

가능하다면 고차원의 희소 벡터에서는 가중치가 정확하게 `0`으로 떨어지도록 유도하는 것이 좋습니다. 가중치가 정확하게 0일 경우 모델에서 해당 특성을 삭제합니다. 특성을 없애면 RAM이 절약되고 모델의 노이즈가 줄어들 수 있습니다.

예를 들어, 캘리포니아뿐만 아니라 전 세계를 포괄하는 주택 데이터 세트를 생각해 보겠습니다. 분 단위(1분 =1/60도)로 전 세계 위도를 버케팅하면 약 10,000개의 차원을, 전 세계 경도를 버케팅하면 약 20,000개의 차원을 희소 인코딩에서 사용할 수 있습니다. 이러한 두 가지 특성에 대한 특성 교차(feature cross)를 통해 약 2억 개의 차원이 생성됩니다. 2억 개의 차원 중 많은 부분이 제한된 거주 지역(예: 바다 한가운데)을 나타내므로 이러한 데이터를 효과적인 일반화를 위해 사용하기는 어려울 것입니다. 이와 같이 불필요한 차원을 저장하기 위해 RAM을 낭비하는 것은 어리석은 일일 것입니다. 따라서 무의미한 차원의 가중치가 정확히 0이 되도록 하는 것이 중요합니다. 그러면 추론 단계에서 이러한 모델 계수(coefficients)에 대한 저장 비용을 지불하지 않아도 됩니다.

적절히 선택한 정규화 항을 추가함으로써, 학습 시 수행한 최적화 문제에 이 아이디어를 적용할 수 있습니다.

L_2 정규화를 통해 목표를 달성할 수 있을까요? 안타깝게도 그렇지 않습니다. L_2 정규화는 가중치를 작은 값으로 유도하지만 정확히 0.0으로 만들지는 못합니다.

이에 대한 대안은 모델에서 0이 아닌 계수(coefficient) 값의 count에 페널티를 주는 정규화 항을 생성해 보는 것입니다. 0이 아닌 계수(coefficient) 값의 count를 늘리는 것이 정당화되는 유일한 경우는 모델의 데이터 적합성을 충분히 확보한 경우입니다. 하지만 이 count-based 접근 방식은 직관적으로는 매력적이긴 하지만 convex optimization problem를 NP-hard인 non-convex optimization problem로 바꿔버리는 단점이 있습니다. 그렇기 때문에, L_0 정규화로 알려진 이 아이디어를 실제로 효과적으로 사용할 수는 없습니다.

하지만 L_0과 비슷하면서 convex하다는 이점이 있어 계산하기에 효율적인 L_1 regularization라는 regularization term이 있습니다. 따라서 L_1 regularization를 사용하여 모델에서 유용하지 않은 많은 계수(coefficients)를 정확히 0이 되도록 유도하여 추론 단계에서 RAM을 절약할 수 있습니다.

L_1 정규화와 L_2 정규화 비교

L_2와 L_1은 서로 다른 방식으로 가중치에 페널티를 줍니다.

- L_2는 *가중치*^{2}에 페널티를 줍니다.
- L_1은 |*가중치*|에 페널티를 줍니다.

결과적으로 L_2와 L_1은 서로 다르게 미분됩니다.

- L_2의 미분계수는 2 * *가중치*입니다.
- L_1의 미분계수는 *k*(가중치와 무관한 값을 갖는 상수)입니다.

L_2의 미분계수는 매번 가중치의 x%만큼 제거한다고 생각하면 됩니다. 그리스 철학자 제논이 말했듯이 어떤 수를 x%만큼 *무한히* 제거해도 그 값은 절대 0이 되지 않습니다. L_2는 일반적으로 가중치를 0으로 유도하지 않습니다.

L_1의 미분계수는 매번 가중치에서 일정 상수를 빼는 것으로 생각하면 됩니다. 하지만 절대값으로 인해 L_1은 0에서 불연속성을 가지며, 이로 인해 0을 지나는 빼기 결과값은 0이 되어 제거됩니다. 예를 들어, 빼기 연산으로 인해 가중치가 +0.1에서 -0.2이 된다면 L_1은 가중치를 정확히 0으로 만들 수 있습니다. 드디어 발견했습니다. L_1을 통해 가중치가 제거되었습니다.

모든 가중치의 절대값에 페널티를 주는 L_1regularization는 다양한 모델에서 아주 효율적으로 활용할 수 있습니다.

이 설명은 1차원 모델에만 적용되는 점에 유의하세요.

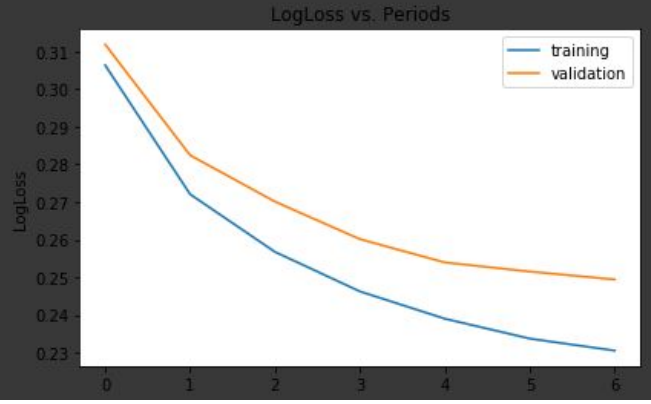
https://colab.research.google.com/notebooks/mlcc/sparsity_and_l1_regularization.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=l1regularization-colab&hl=en#scrollTo=pb7rSrLKIjnS

my_optimizer = tf.train.**FtrlOptimizer**(learning_rate=learning_rate, **l1_regularization_strength=regularization_strengt**h)

https://www.eecs.tufts.edu/%7Edsculley/papers/ad-click-prediction.pdf

```
[11] linear_classifier = train_linear_classifier_model(
         learning_rate=0.1,
         # TWEAK THE REGULARIZATION VALUE BELOW
         regularization_strength=0.0,
         steps=300,
         batch_size=100,
         feature_columns=construct_feature_columns(),
         training_examples=training_examples,
         training_targets=training_targets,
         validation_examples=validation_examples,
         validation_targets=validation_targets)
     print("Model size:", model_size(linear_classifier))
```

```
Training model...
LogLoss (on validation data):
  period 00 : 0.31
  period 01 : 0.28
  period 02 : 0.27
  period 03 : 0.26
  period 04 : 0.25
  period 05 : 0.25
  period 06 : 0.25
Model training finished.
Model size: 788
```




```
[10] linear_classifier = train_linear_classifier_model(
         learning_rate=0.1,
         regularization_strength=0.8,
         steps=300,
         batch_size=100,
         feature_columns=construct_feature_columns(),
         training_examples=training_examples,
         training_targets=training_targets,
         validation_examples=validation_examples,
         validation_targets=validation_targets)
     print("Model size:", model_size(linear_classifier))
```

```
Training model...
LogLoss (on validation data):
  period 00 : 0.34
  period 01 : 0.30
  period 02 : 0.28
  period 03 : 0.27
  period 04 : 0.26
  period 05 : 0.26
  period 06 : 0.25
Model training finished.
Model size: 570
```