Lecture 3: Loss Functions and Optimization

목차

Loss Function

Loss Function 이란 training data에 대하여 score가 얼마나 불만족스러운지를 판단하는 기준이다. 이러한 Loss 값을 낮추기 위하여 parameter를 조절하는데 이를 optimization 이라고 한다.

Dataset 의 x_i 를 image라 하고 y_i 를 label이라 하자.

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Dataset

이때 Loss Function을 나타내면 다음과 같다.

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Loss Function

이는 적당한 W 값을 이용하여 Score 값을 구하고 이를 평균낸다.

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Picture 1

Multiclass SVM loss
SVM을 나타내면 다음과 같다.

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

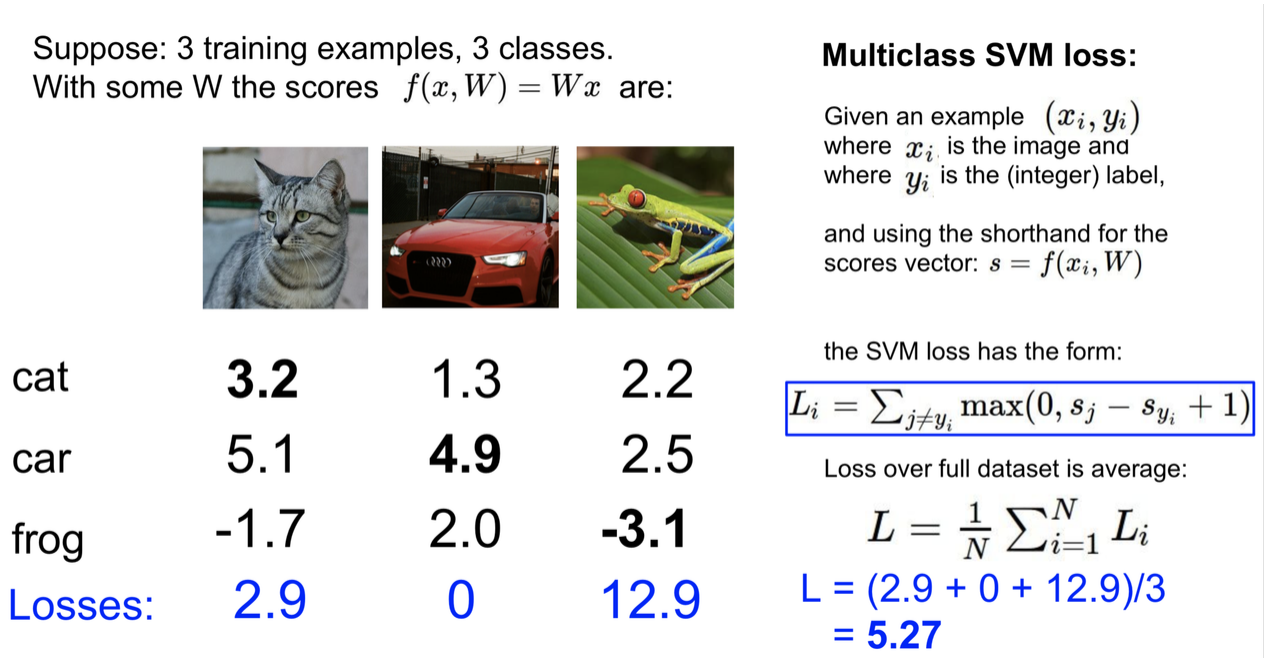
Soft Vector Machine

SVM 은 해당하는 이미지의 score 값을 다른 label 의 score 값과의 차와 bias를 더하였을 때 0보다 큰 값의 합으로 나타낸다.
Picture 1에서 각 이미지별로 L_i 값을 구하면 다음과 같다.
L_1 = Max(5.1 - 3.2 + 1.0) + Max(-1.7 - 3.2 + 1.0) = 2.9 + 0 =2.9
L_2 = Max(1.3 - 4.9 + 1.0) + Max(2.0 - 4.9 + 1.0) = 0 + 0 = 0
L_3 = Max(2.2 + 3.1 + 1.0) + Max(2.5 + 3.1 + 1.0) = 6.3 + 6.6 = 12.9

L_i의 값이 커질수록 손실이 많이 일어났다는 의미이다.

여기서 bias로 1을 두고 있는데 이는 만약에 S_j와 S_y_i의 값이 같으면 Loss가 0이 나오기 때문에 이를 방지하기 위하여 설정한 값이다.
이는 정답인 클래스와 각 클라스의 차이의 정도를 어느정도 이상이 되야하는지를 나타낸다고 볼 수 있다.

위의 결과로 L의 값은
L = 1/3 * (2.9 + 0 + 12.9) = 5.27 이 된다.



Slide

Code

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

SVM Loss

```
# x : images data
# y : label data
def L_i_vectorized(x, y, W):
  scores = W.dot(x)
```

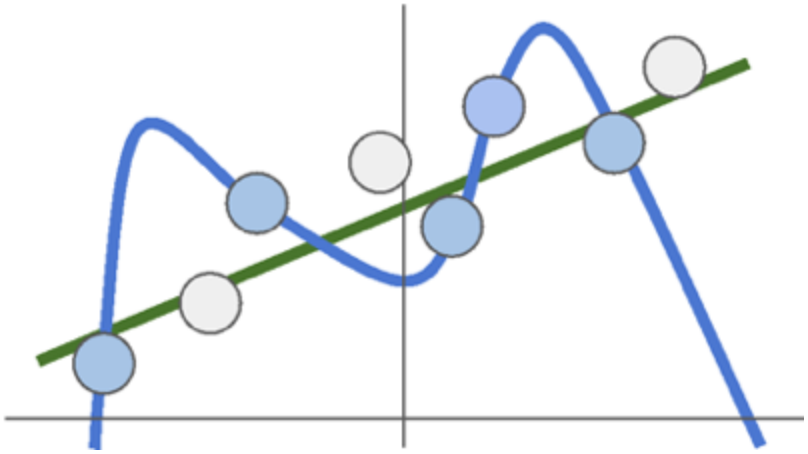
```
# [3.2  1.3  2.2]
# [5.1  4.9  2.5]
# [-1.7 2.0 -3.1]

# scores 값에서 현재 스코어에 해당되는 다 빼준다.
  margins = np.maximum(0, scores - scores[y] + 1)
# [0    0  6.3]
# [2.9  0  6.6]
# [0    0    0]
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i
```

~~왜 margins[y] = 0이라고 설정한지 모르겠다.~~

Regularization

그런데 Data Loss function은 실제 트레이닝 데이터에 적합하게 되어있지만 우리는 test data (또는 validation data)로 검증을 해야한다. test data 로만 모델을 만들 경우 모델의 형태가 복잡해질 수 있다. (고차원 함수가 될 수 있다는 의미) 이러한 복잡한 형태는 train data 에서만 fit 되지 test data 에서는 fit 되지 않을 수 있는데 이를 방지하고자 simple 하게 만들어주는 기법이 Regularization 기법이다.

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data Loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$



흰색 데이터는 test data

Cross-entropy loss (Softmax)

다음 소개되는 내용은 Softmax function이다.
강의 내용에서 소개되기로는 SVM을 이용하여 단순히 Score로 표현하는 방식에서 score를 probability로 표현한다.

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

cat	**3.2**
car	5.1
frog	-1.7

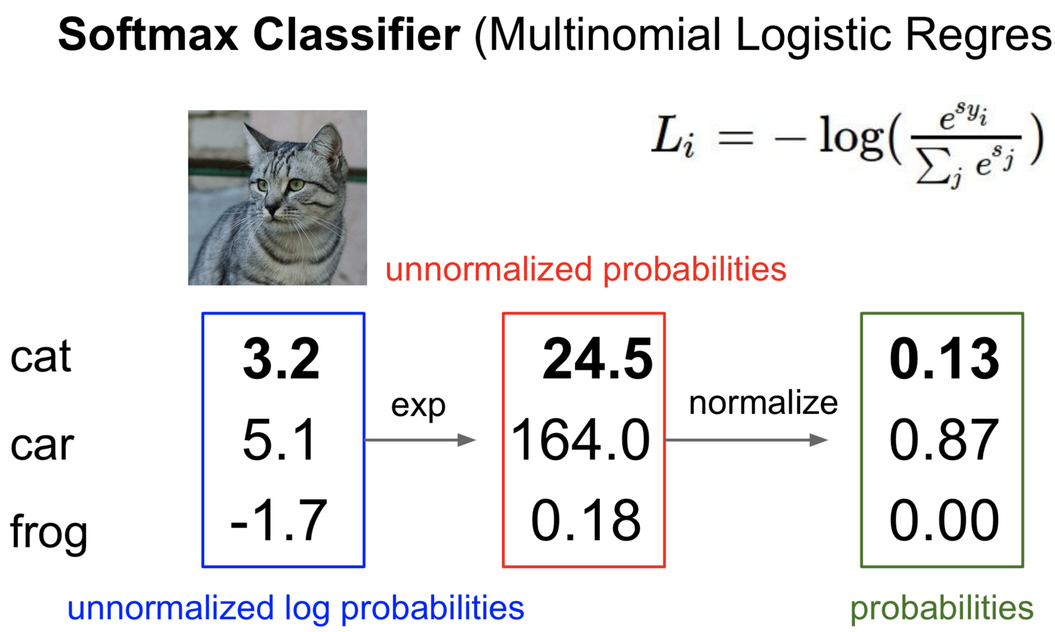
Slide

위의 슬라이드에서 나오는 식과 같이 score 값을 probability로 표현해준다. 이는 각 score 값의 차이를 크게 하여 이를 정규화하는 것으로 나타낼 수 있다.
이때 softmax function을 통과한 score 값은 0 ~ 1 의 값을 가지며 합은 총 합은 1이 된다.
이를 이용하여 Loss function을 표현하면 다음과 같다.

$$L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$$

softmax loss function

이는 지수화한 score 값을 log로 차원을 낮춰주고 0 ~ 1의 값을 가지므로 -를 취해줘서 양의 실수로 표현해준다.



-log(0.13) = 0.89

Softmax vs SVM

Softmax vs. SVM

$$L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}}) \qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

---

assume scores:
[10, -2, 3]
[10, 9, 9]
[10, -100, -100]
and $y_i = 0$

Q: Suppose I take a datapoint and I jiggle a bit (changing its score slightly). What happens to the loss in both cases?

만약 score 값에 변화를 주었을때 각각의 경우 어떠할지 보자.
SVM의 경우 정답의 클래스의 score 값을 크게 하였을 때는 어차피 Max(0, s_j - s_y_i + 1) 로 계산하기 때문에 기존의 L_i 가 0에서 변화를 하지 않을 수 있다. 즉 변화에 둔하다.
하지만 softmax 는 각 스코어의 차이를 크게 만들었기 때문에 변화에 민감하다.

Recap

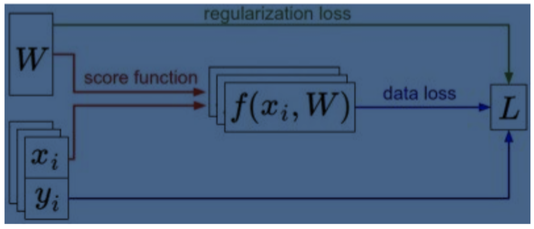
How do we find the best W?

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax
$$L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$$

SVM
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$ Full loss



이렇게 우리는 W 값을 이용하여 Loss 를 구하고 Regularization 을 구하는 방법을 알아보았다.
그러면 W를 어떻게 구하는지 이제 알아볼 차례이다.

Optimization

우리는 우선 loss 값이 가장 적게 유도되는 W를 찾을 것이다.
이는 랜덤한 W 값을 이용하여 찾아낸다.

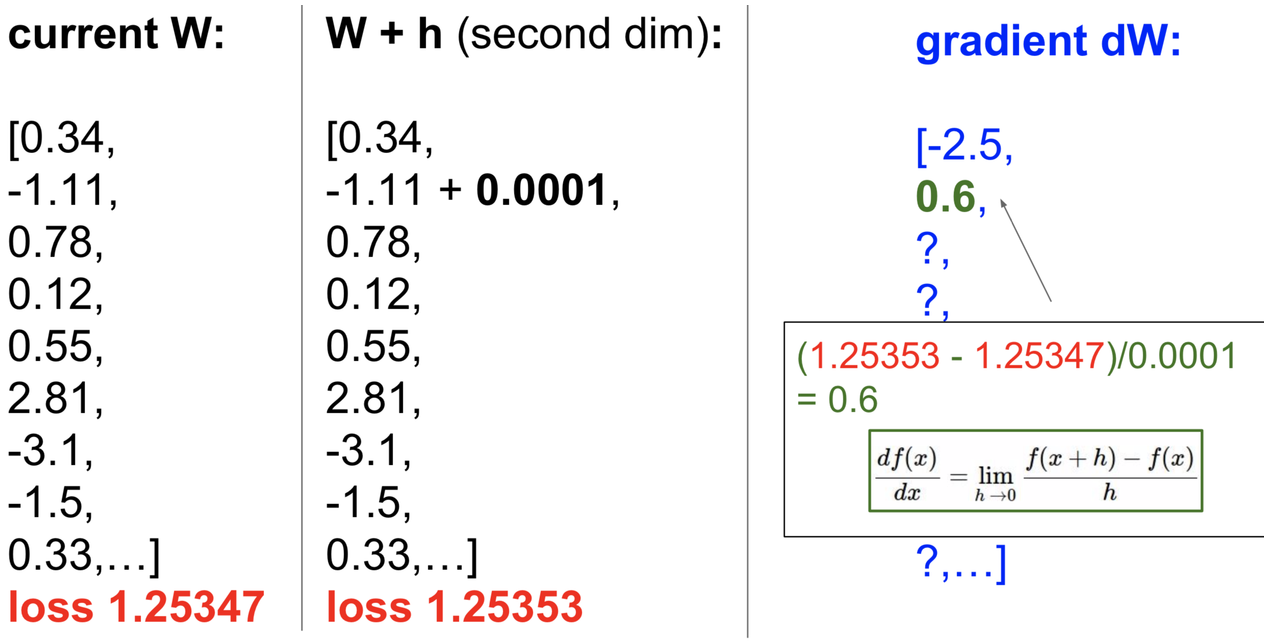
Strategy #1: A first very bad idea solution: **Random search**

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (truncated: continues for 1000 lines)
```

정확도 15.5%가 나온다고 한다. (랜덤확률 10%보단 크다)

Gradient Descent

두 번째 방법은 Gradient Descent 방법을 이용한 것이다. 이는 현재 보고있는 W 의 값을 통해서 얻은 Loss와 W + h 의 값을 통해 얻는 Loss 값의 차를 h 로 나누어준 값, 즉 현재 W에서 미분한 값 dW를 구하는 것이다.

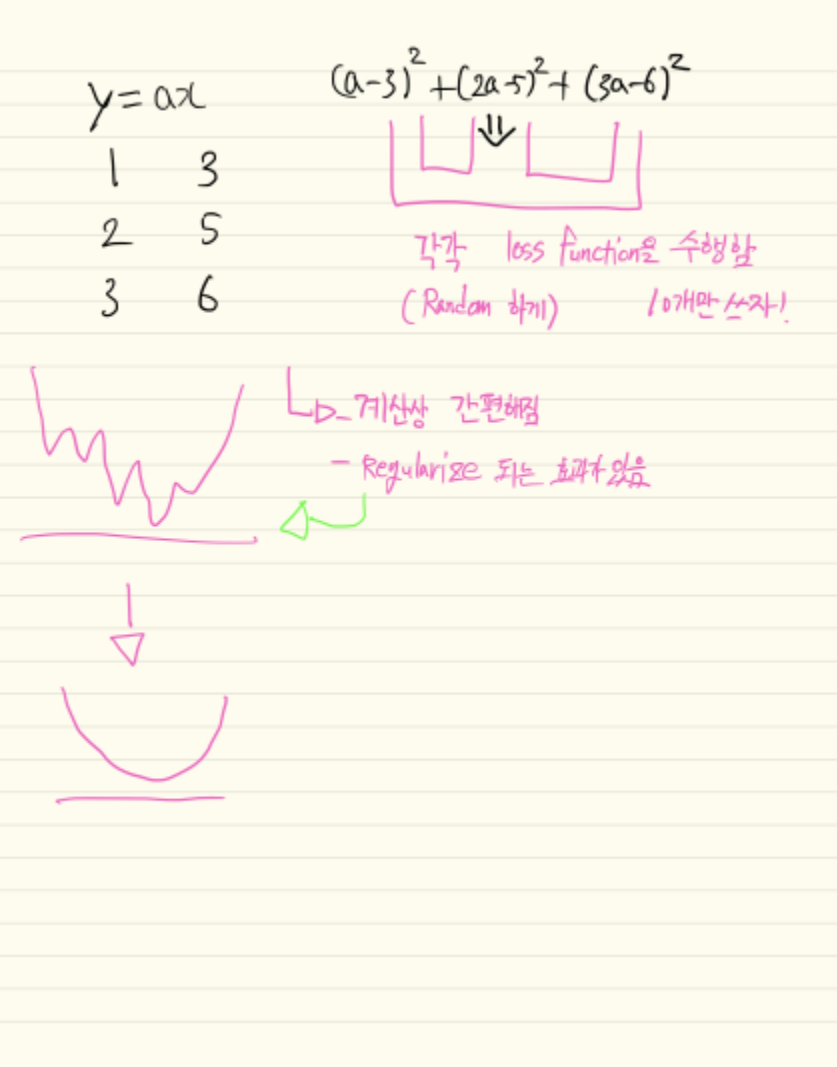


Gradient Descent

하지만 이 방법은 복잡한 Loss function을 모든 case에 대하여 iterative 하게 구하게 되므로 매우 느린 방법이 된다.
그런데 Loss function 은 just function of W 이므로 **Calculus** 를 이용하면 보다 간단하게 구할 수 있다.

Stochastic Gradient Descent

Stochastic 한 Gradient Descent는 입력갯수 N이 너무 많으므로 어느정도 줄여서 W를 구하는 방식이다. 이러한 방식을 이용하면 Model 이 어느정도 정규화되는 모습을 볼 수 있다.



Stochastic Gradient Descent

Jun's Note

1. 무엇을 배웠는지
   > *W를 이용하여 Loss function을 이용해 정답을 찾는데 있어서 어느정도 손실이 발생했는지 알아보는 Loss function을 배웠고 Loss를 최소화하는 W를 구하기 위한 dW를 구하는 법을 배웠다. 그런데 정작 dW는 구하였으면서 이를 이용해 W를 구하는 법은 배우지 않았다. 이후에 배우겠지*
2. 어디에 쓰이는지
   > *Loss function : W를 이용해 정답을 구하는데 있어서 어느정도 손실이 발생했는지.*
   > *dW : W를 최소화하는 방향으로 구할때 필요함*
3. 수식의 목적을 잘 파악했는지?
   > *완벽하게 이해함 굳*