Lec3. Loss Functions and Optimization

2 강에서 배웠던 Linear Classifier 는 결과가 그다지 정확하지 않았다. weight 가 좋은지 정량화하는 어떤 것이 필요할 것이다. 최적화란, 그나마 덜 나쁜 w 를 찾는 것이며, 이를 확인하는 척도를 Loss function 을 통해 알아낸다.

Loss Function: classifier 가 어떻게 좋은지를 알려줌

$\{(x_i, y_i)\}_{i=1}^{N}$ 예시 dataset 이 주어졌을 때,

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

전체 Loss 는 각 Loss 의 합

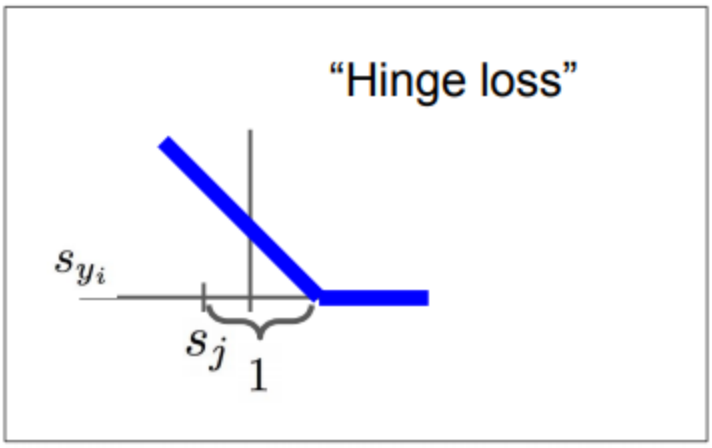
Multiclass SVM(Support Vector Machine) Loss

1. True 인 카테고리를 제외한 나머지 카테고리 Y 의 합을 구함(incorrect class 를 전부 합함)
2. 올바른 카테고리의 스코어와 올바르지 않은 카테고리의 스코어 비교
3. 올바른 클래스의 점수가 올바르지 않은 클래스의 점수보다 높으면, 그 격차가 safety margin 이상이라면 True 인 스코어가 다른 False 클래스보다 훨씬 크다는 의미 => 이 때 Loss 는 0 이 됨 (해당 예시에서는 safety margin 을 1 로 잡음)

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

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

form of SVM loss



위의 그래프에서 y 축은 Loss, x 축은 s_yi 를 의미함.

- correct class 의 score 가 올라갈수록 loss 가 선형적으로 줄어드는 모습을 볼 수 있음
- Loss 가 0 이라는 것은 class 를 잘 분류했음을 의미함
- correct score 가 incorrect score 보다 높으면 좋음
- correct score 는 saftey margin 을 두고 다른 다른 socre 보다 훨씬 더 높아야 함 => 충분히 높지 않으면 Loss 가 커짐

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

SVM loss 를 간단히 쓰면 위의 수식과 같음
- 시그마를 확인하면, correct class 가 아닌 class 를 순회함을 볼 수 있음
full dataset 의 loss 는 아래의 식처럼 각 dataset loss 의 평균임

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

Q1: 만약 한 class 의 score 가 조금 바뀌면 어떻게 될까?
A1: SVM loss 는 correct score 와 incorrect score 의 차이를 고려하기 때문에 Loss 자체는 변하지 않을 것
Q2: loss 의 min/max 는?
A2: min 은 0. 모든 클래스에 걸쳐서 correct class 의 score 가 제일 큰 경우임. max 는 ∞. correct score 가 엄청 작은 음수인 경우
Q3: w 의 초기값이 작아서 s 가 0 에 수렴하면 loss 는 어떻게 될까?
A3: 클래스의 수 - 1
Q4: 합이 정답 클래스까지 포함해서 진행되면 어떻게 되는가?
A4:Loss 가 safety margin 만큼 증가함
Q5: 합 대신에 평균이 사용되면 어떻게 변하는가?
A5: scale 만 변할 뿐, loss 에 큰 영향을 미치지는 않음
Q6: 제곱을 취한다면?
A6: 결과는 달라짐. good and bad 의 trade off 를 비선형적 방식으로 바꿔주어 손실함수의 계산 자체가 바뀜. 실제로도 차이를 극명하게 보여주는 Squared Hinge Loss 를 종종사용함. 손실함수는 어떤 에러를 내가 신경쓰고 있는지, 어떤 에러가 trade off 되는지를 보여주는 척도이므로 필요에 따라 손실함수를 잘 설계하는 것이 필요함.

```
def L_i_vectorized(x, y, W):
  scores = W.dot(x)
  margins = np.maximum(0, scores - scores[y] + 1)
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i
```

SVM Loss example code

Q: 만약 L = 0 이 되는 W 를 찾았을 때, 그 W 는 유일한가?
A: 그렇지 않음. 다른 W 도 존재할 수 있음 (2W 도 L = 0 일 것. W 의 스케일은 변함)

우리는 training dataset 에 대한 classifier 의 성능에 관심이 있는 게 아니라 test data 에 대한 classifier 의 성능에 관심이 있음. 죽, training dataset 의 Loss 에만 신경을 써서는 안됨. 이 때문에 추가된 것이 Regularization term 임

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

Regularization term: 모델이 좀 더 단순한 w 를 선택하도록 도와주는 부분.
- 모델이 더 복잡해지지 않게 함
- soft penalty 를 추가함

+를 기점으로 우항의 왼쪽 부분은 data loss term, 오른쪽 부분은 regularization term 임
regularization term 에서 람다는 두 항간의 trade off 를 의미함(hyperparameter)

Regularization: model 이 training data set 에 완벽히 fit 하지 못하도록 모델의 복잡도에 패널티를 부여하는 방법
- L2 regularization(= Euclidean Norm): 가중치 행렬 W 에 대한 Euclidean norm
- L1 regularization: 행렬 W 가 희소행렬이 되도록 하고, L1 norm 으로 W 에 penalty 를 부과함
- Elastic net: L1 과 L2 를 합친 형태
- Max norm regularization
- Dropout

Example

$$x = [1, 1, 1, 1] \qquad R(W) = \sum_k \sum_l W_{k,l}^2$$

$$w_1 = [1, 0, 0, 0]$$
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

강의에서 든 예시를 보면, 위의 w1 과 w2 는 x 와의 내적이 1 로 동일하므로 Linear classification 의 관점에서 같다고 볼 수 있음.
하지만, L2 regularization 은 norm 이 더 작은 w2 를 더 선호함. 이 방식은 둘 중 어떤 것이 더 coarse 한지 측정함. 즉, x 의 모든 요소가 영향을 줬으면 하는 것이 해당 방식임. 퍼져있으면 덜 복잡하다고 생각함.
L1 regularization 의 경우, w1 을 선호함. 가중치 w 에 있는 0 의 수에 따라 모델의 복잡도를 다루는 방식임. w1, w2 의 L1 의 값은 같음. 하지만, "일반적으로 L1 은 sparse 한 solution 을 선호"함. L1 이 복잡하다고 느끼고 측정하는 것은 0 이 아닌 요소의 개수임.
다시 언급하자면, Linear Classification 에서 W 는 얼마나 x 가 output class 와 닮았는지를 나타내는 것임.

Softmax Classifier(Multinomial Logistic Regression): 해당 loss function 은 score 자체에 추가적인 의미를 부여함

scores = unnormalized log probabilities of the classes.

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

1. score 전부 이용
2. score 에 지수를 취해서 양수로 바꿈

3. 그 지수들의 합으로 다시 정교화
=> 이 함수를 거치면 확률분포를 얻을 수 있고, 이는 해당 class 의 확률이 됨

$$L_i = -\log P(Y = y_i | X = x_i)$$

만약, 이미지가 고양이라면 실제로 고양이일 확률이 1 이고 나머지 class 의 확률은 0 임을 알 수 있음. 즉, softmax 에서 나온 확률에서 correct class 에 해당하는 확률을 1 로 나타내게 해야함. 이를 좀 더 쉽게 찾기 위해 log 를 활용함. log 함수는 단순 증가 함수이므로 그냥 확률값을 최대화시키는 것보다 log 를 최대화시키는 것이 더 쉬움. 그리고 Loss function 은 얼마나 잘 분류하지 못했는지의 척도임으로 음의 방향으로 가게 바꿔야 함. 그러면 위와 같은 식이 나옴. 식을 좀 더 간소화하면 아래와 같음

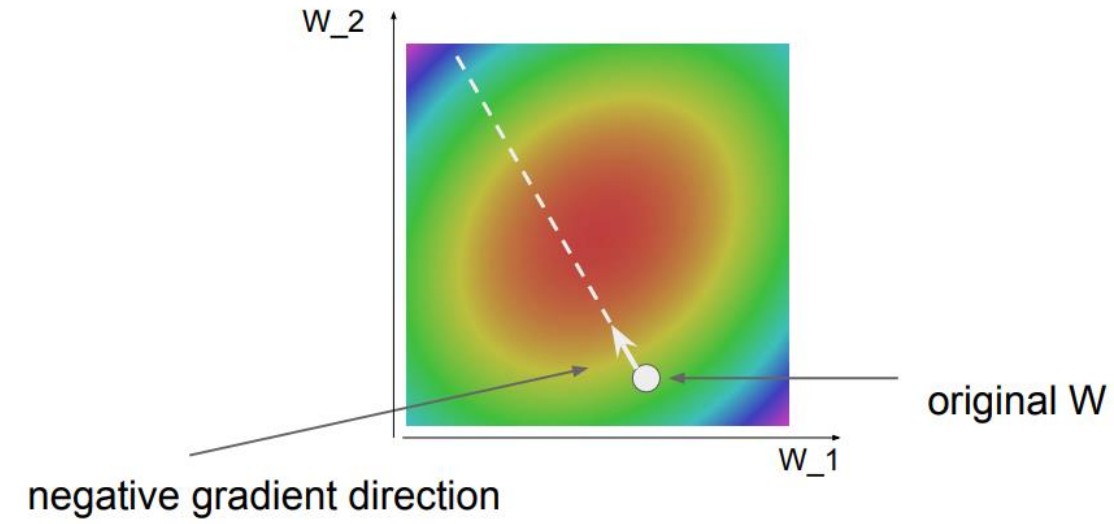
$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

score 가 softmax 를 거치고 거기에 -log 를 취해주면 그것이 loss function 이 됨
Q1: loss 의 min/max 는?
A1: 이론적으로, min: 0. correct score 가 ∞이고, incorrect score 가 -∞여서 log1 = 0 인 경우. 이론적으로, max: ∞. correct class score 가 -∞여서 correct class 의 확률이 0 일 때. 하지만 유한정밀도를 가지고는 앞서 말한 것과 같은 min/max 가 나올 수 없음. 지극히 이론적인 이야기
Q2: w 의 초기값이 작아서 s 가 0 에 수렴하면 loss 는 어떻게 될까?
A2: logC

SVM 과 Softmax 는 대부분 비슷함. 다만, SVM 에서는 correct class 와 incorrect class 의 margin 을 통해 loss 를 확인하고, Softmax 에서는 확률을 구해 -log(correct class)를 통해 loss 를 확인함.
Q: score 를 약간 바꿨을 때 Loss 는 어떻게 되는가?
A: SVM 은 일정 선(margins)에 도달하면 성능 개선에 신경쓰지 않음. Softmax 는 계속 더 성능을 높이려 함. 다만, 실재 딥러닝 어플리케이션에서 두 손실 함수 간의 성능 차이는 엄청나게 크지는 않음

그래서 best W 는 어떻게 구하는가 => 최적화!
1. Random search: 임의로 샘플링한 W 를 엄청 많이 모아놓고 Loss 를 계산해서 어떤 W 가 좋은지 알아보는 방식. 굉장히 비효율적인 방식임
2. Follow the slope: NN 이나 linear classifier 같은 것을 훈련시킬 때 일반적으로 사용. local geometry 를 사용하는 방식.

gradient: 편도함수들의 벡터
- 특정 방향에서 얼마나 가파른지 알고 싶을 때, 그 방향의 unit vector 와 gradient vector 를 내적해 확인함
- 이의 각 요소가 알려주는 것: 우리가 그 점으로 갈 때 함수 f 의 slope 이 어떤지
- gradient 의 방향: 가장 많이 올라가는 방향. 반대 방향은 가장 많이 내려가는 방향
- 함수의 어떤 점에서의 선형 1 차근사 함수를 알려주므로 이는 매우 중요함.
컴퓨터로 gradient 를 이용하는 가장 쉬운 방법: finite difference methods
- 하나하나 계산하는 방식으로 굉장히 느린 방식임.
=>calculus 활용
- W 의 모든 원소를 순회하여 gradient 를 구하고 변화량을 계산하는 것이 아니라 gradient 를 나타내는 식이 뭔지만 먼저 찾아내고 그걸 수식으로 나타내서 한 번에 gradient dW 를 계산함



2 차원 공간의 예시

- bowl 처럼 생긴 것이 손실 함수의 모습임. 빨간쪽으로 갈수록 loss 가 낮은 것을 의미함.
- 임의의 지점에 W 를 설정해두고, -gradient 를 계산하면 결국엔 가장 낮은 지점에 도달할 것.

Loss 를 구할 때 N 이 매우 커지면 Loss 를 계산하는 데에 많은 시간이 걸릴 것. 따라서 실제로 사용하는 것은 아래의 방식임

Stochastic Gradient Descent(SGD): 전체 dataset 의 gradient 와 loss 를 계산하기보다, Minibatch 라는 작은 training sample 집합으로 나눠서 학습시킴. 그리고 minibatch 를 이용해서 loss 전체 합의 추정치와 실재 gradient 의 추정치를 계산하는 것.

- 주로 2 의 승수를 minibatch 로(32/64/128)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

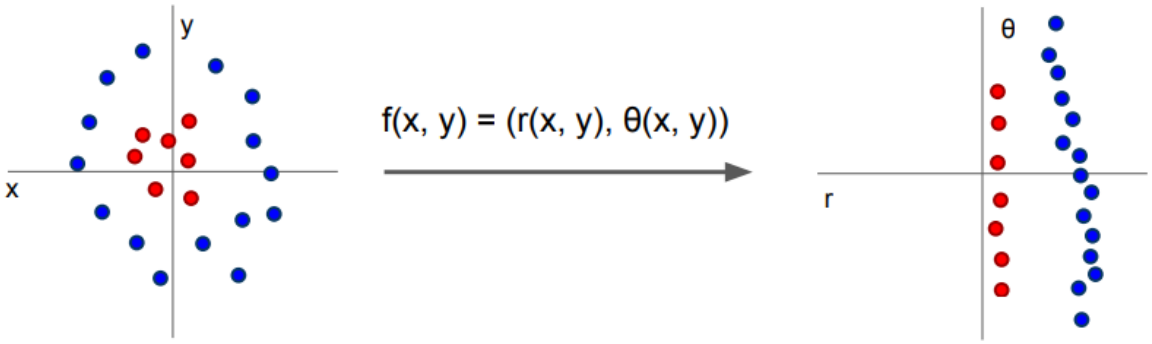
```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

이 링크를 통해 어떤 방식으로 학습이 되는지 확인할 수 있음.

Image Features

DNN 이 유행하기 전에 주로 쓰였던 방법(2 가지 스테이지를 거쳤음)

1. 이미지가 있으면, 여러가지 특징 표현을 계산
2. 여러 특징 표현을 연결해 하나의 특징 벡터로 만듦



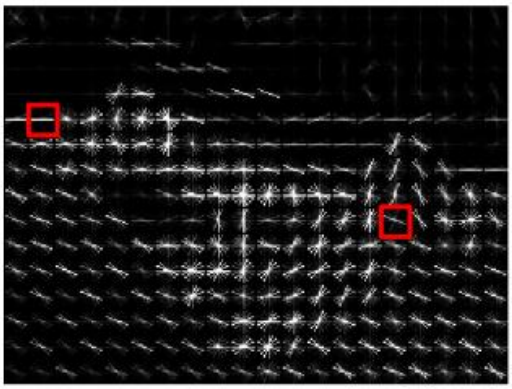
위처럼 linear classifier 로 나눌 수 없었던 점들을 feature transform 을 통해 linear classifier 로 나눌 수 있게 함

Color Histogram: 각 픽셀을 해당하는 색의 바구니에 넣고 각 바구니에 담긴 픽셀의 수를 세는 방식 => 이미지의 전체적인 색을 알려주며, 실제로 사용하는 간단한 특징 벡터임

Histogram of Oriented Gradients(HoG): NN 이 뜨기 전에 인기 있었던 방식

1. 이미지가 있으면 8x8 로 픽셀을 나눔
2. 이 8x8 픽셀 지역 내에서 가장 지배적인 edge 의 방향 계산
3. edge direction 을 양자화해서 바구니에 담음
4. 다양한 edge orientation 에 대한 히스토그램 계산
5. 전체 특징 벡터는 각각의 모든 8x8 픽셀 지역들이 가진 "edge orientation 에 대한 히스토그램"이 되는 것

Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins



Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has 30*40*9 = 10,800 numbers

이 이미지의 경우, 개구리가 앉은 이파리 부분에 대각선 edge 가 많은 것을 볼 수 있음

Bag of Words: 자연어 처리(NLP)에서 영감을 받은 방식

1. 수많은 이미지를 가지고 임의로 조각내고, 그 조각들을 K-means 와 같은 알고리즘으로 군집화함. 이 단계를 거치만 visual word 는 다양한 색과 다양한 방향의 oriented edge 를 포착함.
2. 이미지에서의 visual words 의 발생 빈도를 통해서 이미지를 인코딩함. 이는 이미지가 어떻게 생겼는지에 대한 다양한 정보를 제공함.

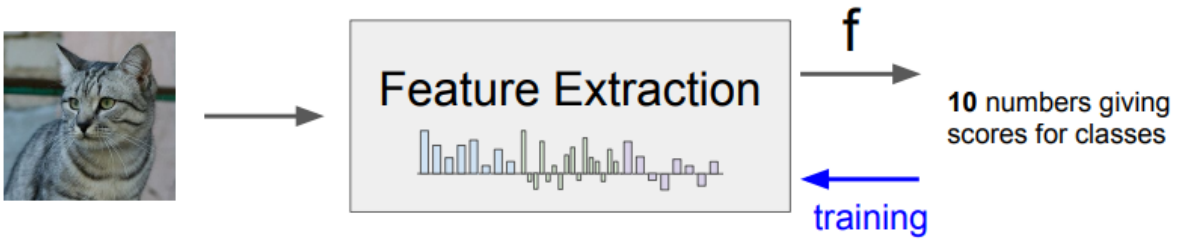
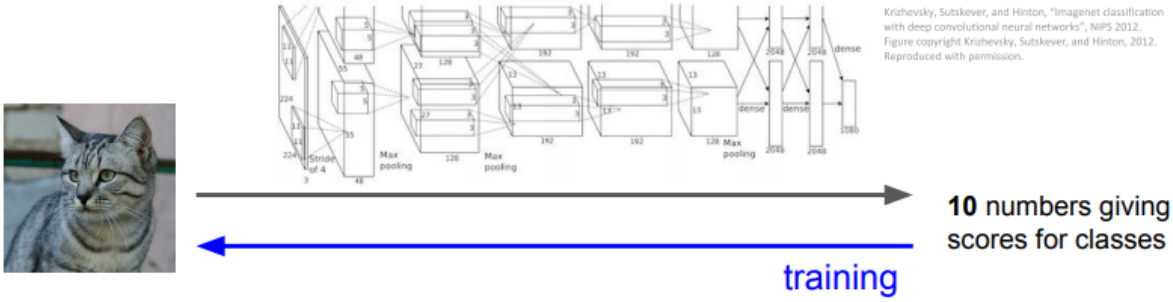


Image feature 의 경우, 특징이 한 번 추출되면 feature extractor 는 classifier 를 training 하는 동안 변하지 않음. 즉, linear classifier 만 훈련함



ConvNets 의 경우, 이미 만들어놓은 특징을 이용하기보다는 데이터로부터 특징들을 직접 학습하게 함. 즉, linear classifier 훈련하고, 가중치 전체를 한꺼번에 학습함

다음 시간에는 neural network 의 개관과 Backpropagation 을 배울 것이다.