

Loss Functions and Optimization

1. 훈련데이터의 학습의 나쁜 정도를 정량화해 학습의 지표로 삼을 수 있는 **loss function** 정의
→ loss를 최소화하는 방향으로 학습해 최적의 파라미터 W 를 찾는 것이 목표
2. loss function을 최소화하는 파라미터(W)를 효율적으로 찾는 방법
⇒ optimization

Loss Function

: 현재 분류기(모델)의 성능이 얼마나 좋은지, 혹은 나쁜지를 알려주는 지표라고 할 수 있음

Loss Function 정리

아래와 같은 데이터셋이 주어졌을 때,

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

(x_i 는 이미지, y_i 는 라벨)

데이터셋의 loss는 샘플들 loss의 합

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

($f(x_i, W)$ 는 예측점수, $L_i()$ 는 loss function)

Loss Function은 이미지 분류 이외의 일반적인 다른 task나 딥러닝 등 input(x)와 label(y)가 있는 어떤 과제에서든지 사용

→ W 의 space에서 loss를 최소화하는 최적의 W 를 찾는 이정표 역할

Muti-class SVM loss

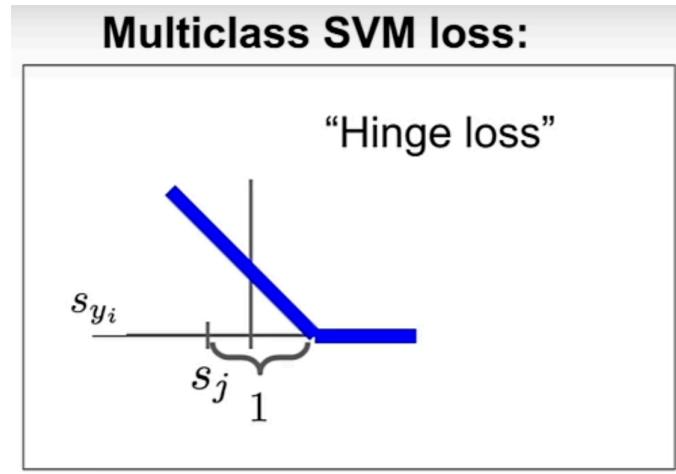
표본 (x_i, y_i) 가 주어지고, (예측)점수 벡터 s 를 $s = f(x_i, W)$ 로 정의해보자.

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

SVM loss는 위와 같은 식을 가진다.

→

s_y 는 실제 라벨값으로 예측할 경우의 점수. 즉, s_{y_i} 는 i 번째 샘플의 정답 클래스일 경우의 점수



위 자료와 같이 x축은 label, y축은 loss라고 설정해보자.

$y = y_{\text{pred}}$ 인 경우 \Rightarrow 즉, 제대로 분류한 경우의 score가 증가할 때 loss가 safety margin에 도달할 때까지 선형적으로 감소하고 이후 0이 됨
 \rightarrow safety margin: 정답 클래스로 분류할 확률이 우세해서 다른 클래스로 분류할 가능성이 매우 희박한 상황

▼ Multi-class SVM loss: 명확한 예시로 설명

<p>Suppose: 3 training examples, 3 classes. With some W the scores $f(x, W) = Wx$ are:</p> <table border="0" style="width: 100%; text-align: center;"> <tr> <td></td> <td></td> <td></td> </tr> </table> <table border="0" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%;">cat</td> <td style="width: 15%;">3.2</td> <td style="width: 15%;">1.3</td> <td style="width: 15%;">2.2</td> </tr> <tr> <td>car</td> <td>5.1</td> <td>4.9</td> <td>2.5</td> </tr> <tr> <td>frog</td> <td>-1.7</td> <td>2.0</td> <td>-3.1</td> </tr> <tr> <td>Losses:</td> <td>2.9</td> <td>0</td> <td>12.9</td> </tr> </table>				cat	3.2	1.3	2.2	car	5.1	4.9	2.5	frog	-1.7	2.0	-3.1	Losses:	2.9	0	12.9	<p>Multiclass SVM loss:</p> <p>Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$</p> <p>the SVM loss has the form: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$</p> <p>Loss over full dataset is average: $L = \frac{1}{N} \sum_{i=1}^N L_i$ $L = (2.9 + 0 + 12.9)/3 = 5.27$</p>
cat	3.2	1.3	2.2																	
car	5.1	4.9	2.5																	
frog	-1.7	2.0	-3.1																	
Losses:	2.9	0	12.9																	

$\max(0, s_j - s_{y_i} + 1)$ 을 통해 각 이미지에 대한 예측 점수를 통해 각 샘플마다 loss를 계산하고 샘플들의 loss를 통해 최종 loss 산출

+) Loss에 선형결합은 결과를 바꾸지 않지만 제곱을 취하는 식의 비선형적 변형은 결과를 바꿀 수 있음. 그러므로 원래 사용하던 loss function과 다른 함수가 되어버림. 제곱을 취하는 변형은 차이를 더 크게 만드는 경향이 있음. 그러므로 linear vs. square는 우리가 다른 클래스의 오류에 대해서 얼마나 볼 것인지 정량화할 때 자주 마주함.

Loss Function은 알고리즘에게 어떤 종류의 오류를 신경쓰고 trade-off 할 것인지 알려주는 것. 그러므로 어떤 과제에 직면하느냐에 따라 어떤 Loss function을 선택할지 다양함.

- Multi-class SVM Loss 예시 코드

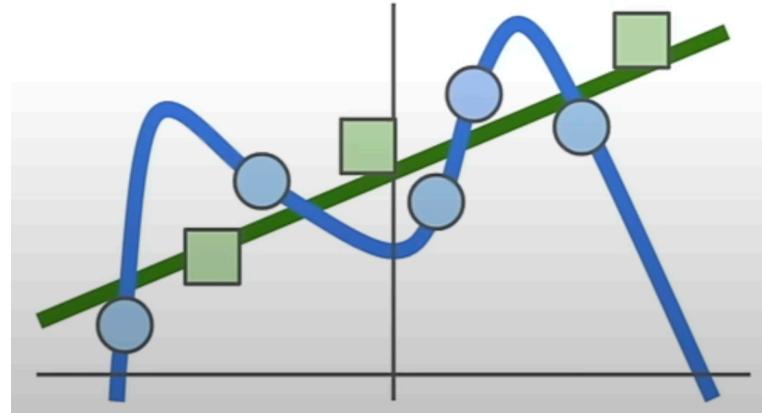
```
def L_i_vectorized(x, y, W):
    scores = W.dot(x)
    margins = np.maximum(0, scores - scores[y] + 1)
    margins[y] = 0
    loss_i = np.sum(margins)
    return loss_i
```

Data Loss

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

: Loss function은 오직 훈련 데이터에 fit한 예측 모델을 학습

But, 우리가 원하는 것은 테스트 데이터에서 예측 성능이 뛰어난 모델



파란 원이 training data, 초록색 사각형이 test data일 때, training data에 fit하게 훈련시킨 파란색 예측 곡선은 test data를 제대로 예측하지 못하는 결과. 오히려 초록색 선이 예측 성능이 높음.

위 문제를 해결하기 위한 방법으로 **Regularization**을 사용

Regularization

: 모델이 복잡하지 않고 단순해야 테스트 데이터에서도 제대로 작동

⇒ 모델의 복잡성을 줄이기 위해 특정한 패널티를 줘 훈련 데이터에 overfitting을 방지하고 일반화 성능 확보하는 방법

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

- **Data Loss term** ⇒ $\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)$
 - **Regularization term** ⇒ $\lambda R(W)$
 - λ ⇒ regularization strength
: Regularization term의 하이퍼파라미터. training data와 test data 사이의 trade-off 관계를 조절하는 역할
- +) Regularization은 고차원의 다항식에서 차수를 줄이는 것과 비슷한 이미지로 이해하면 좋음

Regularization의 종류

L1 regularization

→ 가중치 행렬의 L1 norm에 패널티주는 것

→ 가중치 행렬값 중 영향력이 적은 값이 0이 되는 방향으로 작동 *차원축소 효과

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

L2 regularization

→ 가중치 행렬의 euclidian norm에 패널티주는 것

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

ElasticNet (L1+L2)

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

Max norm regularization

Dropout (esp. DL)

Batch Normalization (esp. DL)

Stochastic depth (esp. DL)

▼ +) L1, L2 regularization은 각각 언제 채택할 것인가?

파라미터 $W \Rightarrow$ 벡터 X 의 값이 결과값에 어떻게 대응되는지에 대한 해석

L1은 영향력이 큰 속성을 크게, 영향력이 작은 속성은 거의 보지 않음

→ (대부분 entry를 0으로 만들기에) 일반적으로 sparse solution을 선호하는 경우 채택

L2는 모든 속성의 값을 고르게 보는 특성

→ 벡터 X 의 모든 값을 잘 퍼뜨리는 경우, 모든 속성의 전체적인 영향력을 보려고 할 때 채택

과제(모델)의 complexity를 어떻게 정의하나에 따라서 어떤 regularization 방식을 채택할지 결정

→ model, data 등을 종합적으로 고려해 선택

- L1의 complexity를 측정하는 방법은 non-zero entry의 수라고 볼 수 있음
- L2는 모든 값에 W 를 퍼뜨리는 것이 덜 complex한 것. 그러므로 non-zero가 less complex

Softmax Classifier (Multinomial Logistic Regression)

Multi-class SVM

→ 각 클래스의 score은 단지 상대적인 숫자로 간주하고 해석이 부족

Softmax

→ 클래스 score에 부가적인 의미 부여

⇒ scores = unnormalized log probabilities of the class

Softmax function

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

위의 Softmax function*을 지나면 probability distribution 생성

* 모든 클래스 score를 거듭제곱해서 양수로 만들고 그 값을 모두 더한 값으로 normalize

- 각 클래스 score은 클래스에 속할 확률인 0에서 1 사이 값을 가짐
- 모든 클래스 score의 합은 1

computed probability distribution와 target probability distribution의 차이를 통해 loss를 계산하고자 함

이때, true class의 log likelihood를 최대화하거나 negative log likelihood를 최소화하는 방식으로 수식을 설정

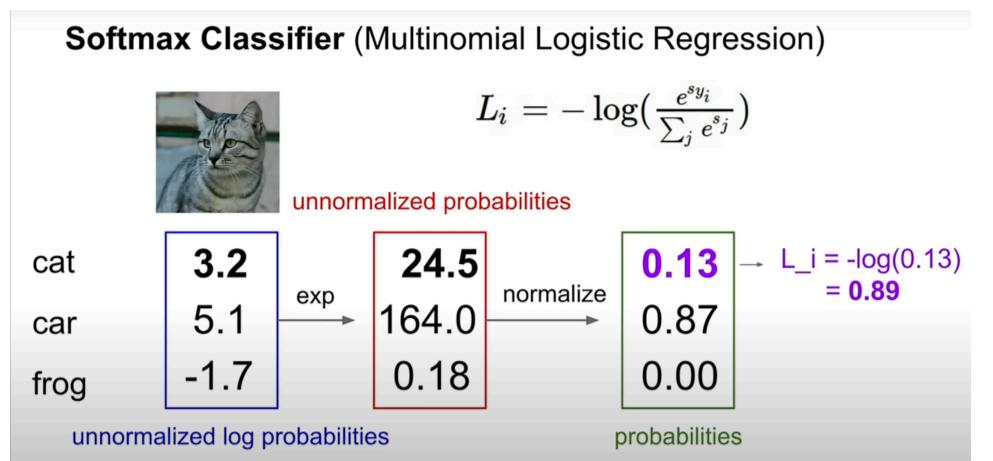
$$L_i = -\log P(Y = y_i | X = x_i)$$

우리가 원하는 것은 badness를 정량화할 loss를 얻는 식이므로 loss function은 true class의 negative log likelihood

정리하면,

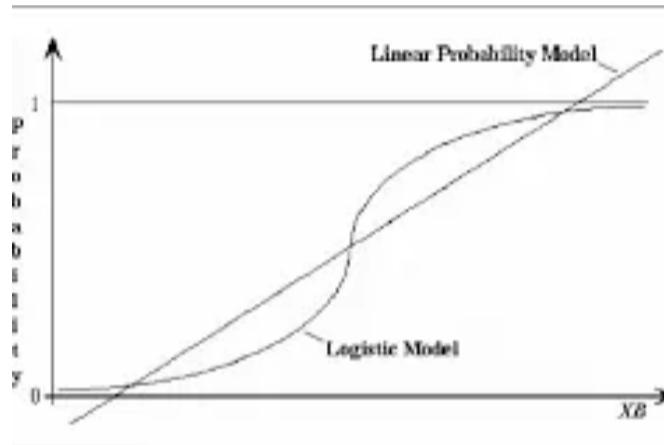
$$L_i = -\log \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$$

▼ Softmax function 예시



▼ 확률에 log를 취하는 이유

raw probability보다 log를 취한 probability를 최대화하는 것이 더 쉬움



Multi-class SVM vs. Softmax

주어진 벡터에서 첫 번째 score인 10이 정답클래스라고 가정하고,

[10, -2, 3], [10, 9, 9], [10, -100, -100]이 주어질 때

- Multi-class SVM: 세 경우 모두 loss가 같음
→ 정답 클래스 점수가 다른 클래스보다 적정 수준 이상이라면 그 이상 관여하지 않음
- Softmax: 세 경우 모두 loss가 다름
→ loss를 계산할 때 모든 데이터 포인트를 점점 개선하기 위해 정답과 다른 클래스 점수와의 차이 수준도 고려

Optimization

: 파라미터 W가 어떻게 loss를 최소화할건지, 그 방법에 대한 것

Strategy #1. Random search

Strategy #2. Follow the slope

Follow the slope

: slope는 함수의 미분

1-dim일 때 함수의 미분은 다음과 같음

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

비슷한 개념으로 다차원일 때 gradient(기울기)는 편미분의 벡터

gradient의 각 element는 해당 좌표 방향으로 움직일 때 function f 의 경사를 나타냄
 ⇒ negative gradient direction을 찾으면 함수(값)를 최소화하는 방향을 찾을 수 있음
 gradient와 방향을 나타내는 유닛 벡터와의 내적을 통해서 slope를 구할 수 있음

위와 같은 방식으로 loss function을 최소화하는 방향으로 파라미터 W 를 업데이트

current W :	$W + h$ (first dim):	gradient dW :
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...] loss 1.25347	[0.34 + 0.0001 , -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...] loss 1.25322	[-2.5, ?, ? $(1.25322 - 1.25347)/0.0001$ $= -2.5$ $\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$?, ?,...]

current W 에 작은 수 h 를 더해 loss 차이를 통해 gradient dW 를 계산. 위에서 봤던 1-dim 함수 미분을 벡터의 element만큼 반복하는 것과 같음

정리

Numerical gradient

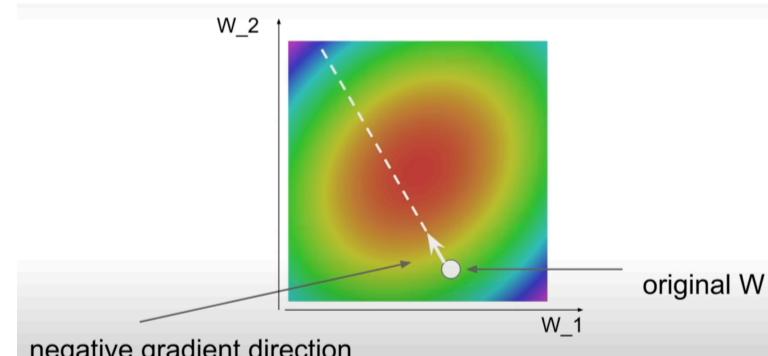
: 정확성이 떨어짐, 느림, 수식으로 나타내기 쉬움

Analytic gradient

: 정확함, 빠름, 오류나기 쉬움

→ 실제로 항상 Analytic gradient 방식을 채용하지만, 디버깅할 때 numerical gradient를 사용. 이 방법을 gradient check라고 함

Gradient Descent



임의의 data point에서 시작해 loss가 가장 적어질 것으로 예상되는 negative gradient direction으로 W 를 업데이트 반복해 region of minima로 향하게 하는 방식

Vanilla Gradient Descent

```
while True:  
    weights_grad = evaluate_gradient(loss_fun, data, weights)  
    weights += - step_size * weight_grad # perform parameter update
```

Gradient Descent는 각각의 모든 훈련 데이터에 대한 loss를 계산(모든 훈련데이터 loss function의 평균)

→ 모든 훈련 데이터 loss를 계산하는 방식은 굉장히 고비용, 비효율적 과정이므로 매우 느림

Stochastic Gradient Descent (SGD)

: 훈련 데이터 수가 방대할 때 미니배치를 사용해서 전체 loss를 추정하는 방식

- 미니배치(minibatch)

: 훈련 데이터셋에서 임의 크기(32, 64, 128, ...)의 작은 데이터 세트를 샘플링한 것

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

```
# Vanilla Minibatch Gradient Descent
```

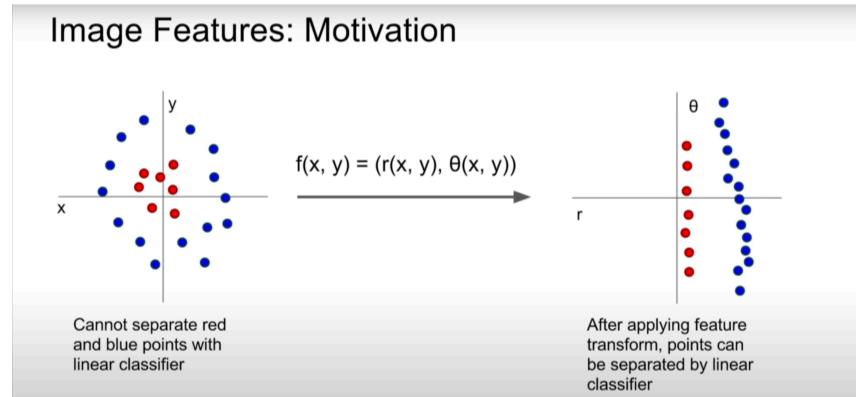
```
while True:
```

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

Image Features

직전 강의에서 언급했듯이 간단한 classifier로 해결할 수 없는 다양한 문제에 직면

→ 이를 해결하기 위해서 feature representation 연구



이미지 데이터같은 경우, raw pixel 데이터를 사용하기보다 feature representation을 추출하면 (새로운 패턴을 발견해) 위와 성능에 도움이 될 수 있음

Feature representation

Examples: Color Histogram

- Hue color spectrum으로 각 픽셀의 색을 추출해 전체적인 이미지가 어떤 색상을 띠는지 feature vector 추출

Example: Histogram of Oriented Gradients (HoG)

- 이미지의 픽셀을 작은 정사각형 구역으로 나눠 이미지의 주요 모서리 방향을 정량화하는 방식으로 feature vector 추축

Example: Bag of Words

- 이미지를 무선적으로 잘라 K-means와 같은 클러스터링 방식으로 code book을 생성한 후, 이미지를 인코드

Image features vs. ConvNets

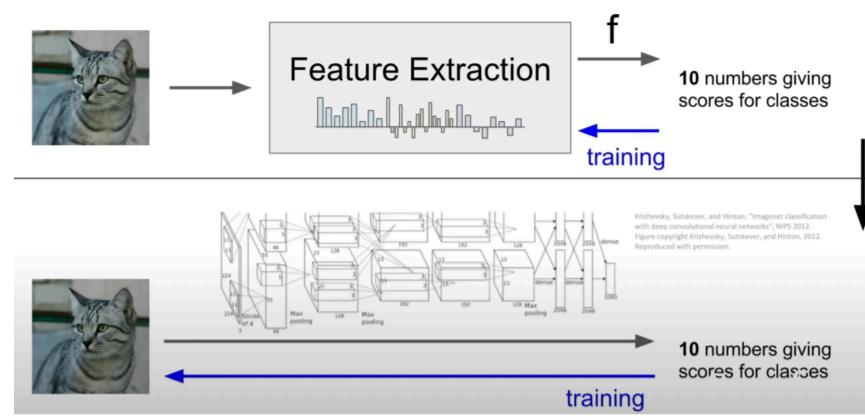
Image features

: 사전에 feature 정의

ConvNets

: 데이터에서 feature 학습

- raw pixels를 입력으로 ConvNetsdp 넣어 수많은 다양한 layers를 통해 계산 (=feature representation). 이후 그 데이터를 기반으로 전체 모델의 모든 파라미터 W 학습



과정은 크게 다르지 않지만, Feature Extraction이 사전에 feature을 정의하는 것과 다르게 ConvNets은 데이터에서 feature을 학습하는 방식