

Week3

✓ 예습	미완료
✓ 복습	미완료
복습과제 날짜	@2022년 9월 12일
예습과제 날짜	@2022년 9월 12일
내용	cs231n Lecture03

강의 요약

저번 시간 복습

TODO

Loss Function

Multiclass SVM loss

Regularization

Softmax Classifier (Multinomial Logistic Regression)

SVM and Softmax

Summary

Optimization

Random search

Follow the slope

Gradient Descent

Stochastic Gradient Descent (SGD)

기존 이미지 추출방식

강의 요약

저번 시간 복습

challenge of recognition, → idea on data driven approach

image classification

분류를 어렵게 하는 조명의 변화, tile, deformation 등을 알아보았다.

KNN - 가장 쉬운 data driven approach (CIFAR-10)

cross-validation, hyperparameter

Linear classifier : parametric classifier

정보를 W 에 저장해서 사용

$Wx + b = y$ (y 는 클래스 점수의 행렬)

“linear decision boundary”를 만들어준다.

TODO

스코어를 보면 classifier가 제대로 작동하고 있지 않음을 알 수 있다.

1. 어떤 W 가 가장 좋을지 정량화하는 방법이 필요하다. → 얼마나 W 가 안 좋은지 알 수 있는 방법이 필요함

⇒ Loss function을 사용

2. W 가 되는 모든 경우에서 가장 least bad한 W 를 찾는 것

⇒ optimization

Loss Function

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

A **loss function** tells how good our current classifier is

Given a dataset of examples

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

Where x_i is image and y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

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

각 트레이닝 샘플의 로스는 해당 트레이닝 샘플을 얼마나 나쁘게 예측하는지를 정량화한다.

전체의 L 은 그러한 샘플의 로스를 모두 더한 값의 평균

딥러닝 알고리즘의 generic set up:

x, y 가 존재, w 가 얼마나 좋은지 정량화하는 함수 L , L 을 최소화 하는 W 를 발견

Multiclass SVM loss

loss function의 일종

x_i 는 이미지, y_i 는 라벨(정수)

점수 벡터 : $s = f(x_i, W)$

SVM loss :

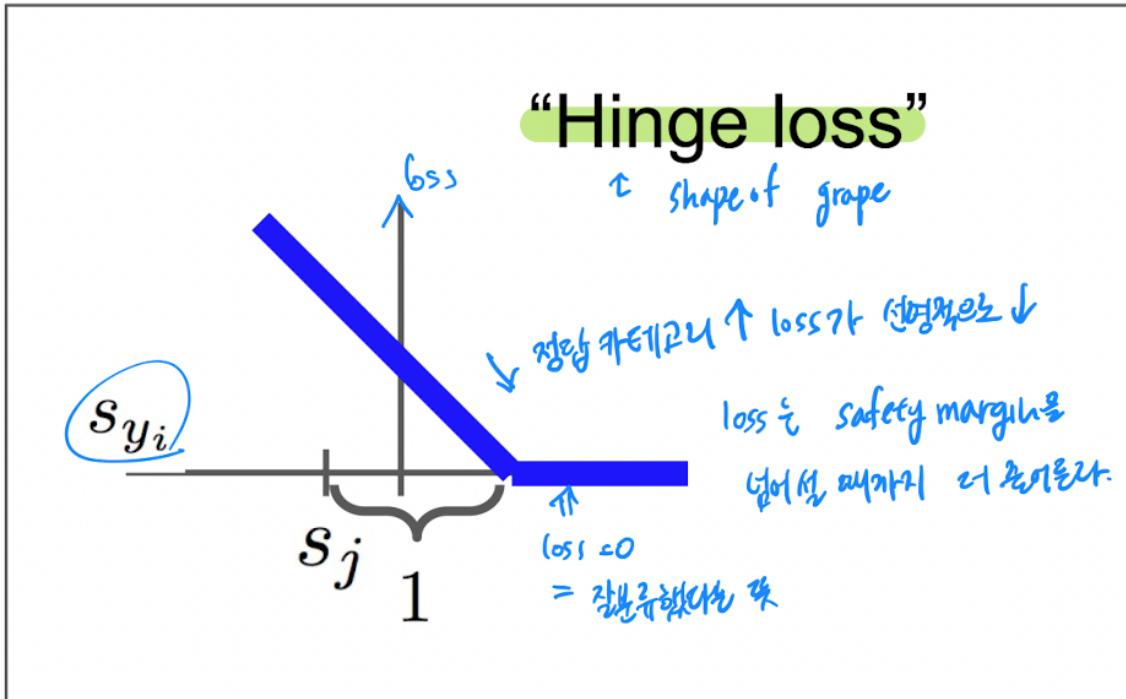
$$\begin{aligned} L_i &= \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases} \\ &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \end{aligned}$$

'true' 카테고리를 제외한 나머지 카테고리와 'true' 카테고리의 점수의 차를 비교.

```
if correct_score - incorrect_score > 1
    loss = 0
else
    loss = incorrect_score - correct_score + 1
```

각 카테고리의 로스를 모두 더한 것이 로스의 합이 된다.

Multiclass SVM loss:



그래프 모양때문에 Hinge Loss라고 불린다. loss가 0에 가까울 수록 잘 분류했다는 뜻이 된다

예시)

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
Losses:			2.9

Multiclass SVM loss:

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: $s = f(x_i, W)$

the SVM loss has the form:

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

$$= \max(0, 5.1 - 3.2 + 1)$$

$$+ \max(0, -1.7 - 3.2 + 1)$$

$$= \max(0, 2.9) + \max(0, -3.9)$$

$$= 2.9 + 0$$

$$= 2.9$$

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
Losses:			2.9

Multiclass SVM loss:

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: $s = f(x_i, W)$

the SVM loss has the form:

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

$$= \max(0, 1.3 - 4.9 + 1)$$

$$+ \max(0, 2.0 - 4.9 + 1)$$

$$= \max(0, -2.6) + \max(0, -1.9)$$

$$= 0 + 0$$

$$= 0$$

전체 로스: 각 로스의 평균

- What happens to loss if car scores change a bit?
 - Car의 점수가 이미 크기 때문에 조금 변경된다고 해서 문제 생기지 않는다. Margin will retain

- What is the min/max possible loss?
 - $\min = 0, \max = \text{infinity}$
- At initialization W is small so all $s \sim 0$. (score are approximately equal) what is the loss?
 - #of class - 1
 - 로스를 계산할 때 정답이 아닌 클래스를 순회한다 ($c-1$ 개).
다 비슷하면 마진 때문에 모두 1을 가리케 된다. $\rightarrow c-1$ 이 된다.
 - 보통 행렬 W 는 임의의 작은 수로 초기화하는데, 처음 학습시에는 결과 스코어가 all uniform random value를 가지게 된다. 유용한 Debugging Strategy가 될 수 있다.
알고 시작하면 loss가 어떻게 되는지 짐작 가능하다.
- What if the sum was over all classes? (including $j = y_i$)
 - loss increases by 1
 - 관례상 정답 클래스 빼고 계산하면 $\minLoss = 0$ 이 된다.

cat	3.2
car	5.1
frog	-1.7
Losses:	2.9

loss increases by 1
 관례상 정답 클래스 빼고 계산하면
 $\minLoss = 0$ 이됨.

$$\begin{aligned}
 & 1 \\
 & + \max(0, 3.2 - 3.2 + 1) \leftarrow \text{정답이 등장} \\
 & + \max(0, 5.1 - 3.2 + 1) \\
 & + \max(0, -1.7 - 3.2 + 1) \\
 & = \max(0, 1) + \max(0, 2.9) + \max(0, -3.9) \\
 & = 1 + 2.9 + 0 = 3.9
 \end{aligned}$$

- What if we used mean instead of sum?
 - 영향을 미치지 않는다.
 - #of class is fixed, mean = rescaling whole function이기 때문에 상관이 없다.
true value of score은 상관이 없다.
- what if we used squared value instead of sum?

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

- 결과가 변경이 된다.
- Changing the trade offs between good and badness in come of non-lined way → 손실 함수 계산 자체가 변경이 된다.
- 제곱합은 실제로 종종 사용되는 손실함수 계산법이다.
- 제곱합을 고려하는 이유는? 잘못 분류되는 것을 막기 위한 것. Hinge Loss에서 조금 차이나는 것이 squared value에서는 많이 차이난다.

Multiclass SVM Loss : Example code

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

```
def L_i_vectorized(x, y, W):
    scores = W.dot(x)
    margins = np.maximum(0, scores - scores[y] + 1)
    margins[y] = 0 # 정답을 가진 샘플은 항상 1이어야 한다.
    loss_i = np.sum(margins)
    return loss_i
```

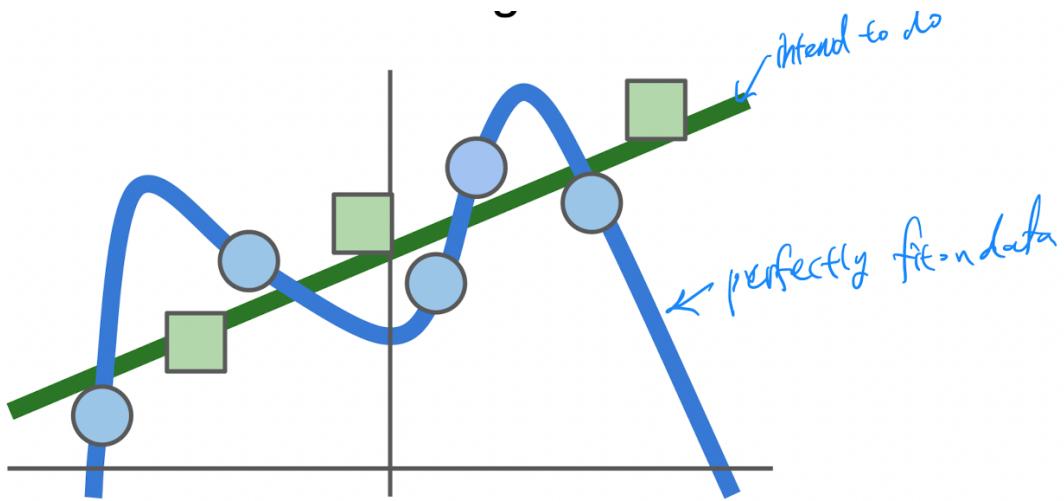
W is unique?

W는 유일한 값이 아니다. 2W도 4W도 L= 0 이다.

Loss function이 classifier에 어떤 W를 찾고 있고, 어떤 W에 신경쓰는지 알려주는 거라면, 다양한 W 중 L=0인걸 선택하는 것은 모순된다.

Only loss in terms of data seems tall classifier to find W that fits to training data. But finning on training data는 크게 신경 쓰지 않아도 된다. training data가 아니라 test data를 잘 분류하는지가 더욱 중요하다.

training data 에 L=0라면 테스트 데이터에 안맞을 수 있다.



Regularization

regularization은 train data에만 맞고, test data에는 맞지 않는 것을 해결해준다.

단순한 W 를 택하도록 도와준다.

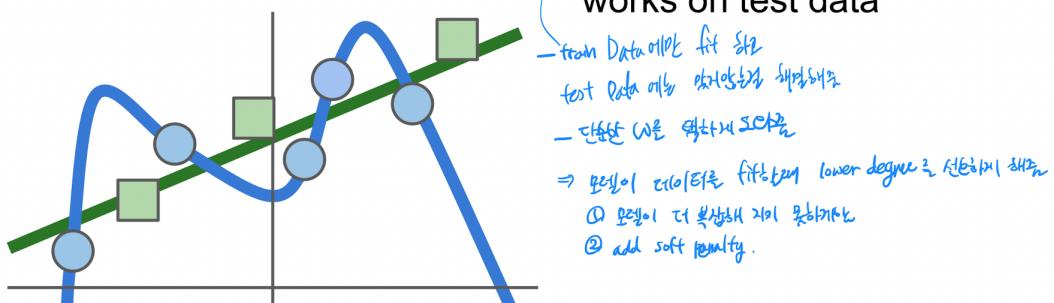
⇒ 모델이 데이터를 fit할 때 lower degree를 선호하게 해준다.

1. 모델이 더 복잡해지지 못하게 한다
2. add soft penalty

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda \underbrace{R(W)}_{\text{Regularization: Model should be "simple", so it works on test data}}$$

Data loss: Model predictions should match training data

Regularization: Model should be “simple”, so it works on test data



- lamda = hyperparameter, regularization strength

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Max norm regularization (might see later)

Dropout (will see later)

Fancier: Batch normalization, stochastic depth

L2 regularization이 가장 많이 사용된다. (= Euclidean Norm, Squared Norm, 1/2 squared norm) : W vector의 euclidean norm에 패널티를 주는 방식

L1 regularization : W가 최소 행렬이 되도록 한다. increasing sparcity in Matrix W

L2 Regularization(Weight Decay)

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

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

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Linear classification을 하면 w1과 w2는 같은 값을 가진다. (내적이 같다)

- L2 Regularization 이 선호하는 것은 w2이다. 왜냐하면 w2가 작은 norm을 가진다. regularization measure 분류기의 복잡도는 relatively coarse way와 관련된다. 즉, x의 모든 요소가 영향을 주었으면 한다.

- L1 Regularization은 L2와 정반대를 선호한다. L1은 W의 개수가 많아질 수록 complexity가 낮아진다.(= prefer sparse solution)

Softmax Classifier (Multinomial Logistic Regression)



scores = unnormalized log probabilities of the classes.

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

high class의 확률은 확률로 나누어 다른 클래스 확률과 합으로 나누어지는 것이다.

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

cat **3.2**

car **5.1**

frog **-1.7**

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

로그는 monotonic function이다. log를 최대화 시키는 것이 그냥 확률 값을 최대화 시키는 것 보다 쉬워서 log를 사용한다.

in summary: $L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$

정답이나 예상과 같은 확률이 높을수록 loss function은 낮아지게 된다.

SVM과 다르게 score에 의미를 둔다.

P를 통해 각 클래스가 될 수 있는 확률을 나열한다.

로그는 monotonic function이다. log를 최대화 시키는 것이 그냥 확률 값을 최대화 시키는 것 보다 쉬워서 log를 사용한다.

loss function은 얼마나 나쁜지를 측정하기 때문에 -를 붙이고 계산한다.

예시)



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

unnormalized probabilities

cat

3.2

\exp

24.5

\rightarrow normalize $\frac{24.5}{164.0}$

0.13

$\rightarrow L_i = -\log(0.13) = 0.89$

car

5.1

164.0

0.87

frog

-1.7

0.18

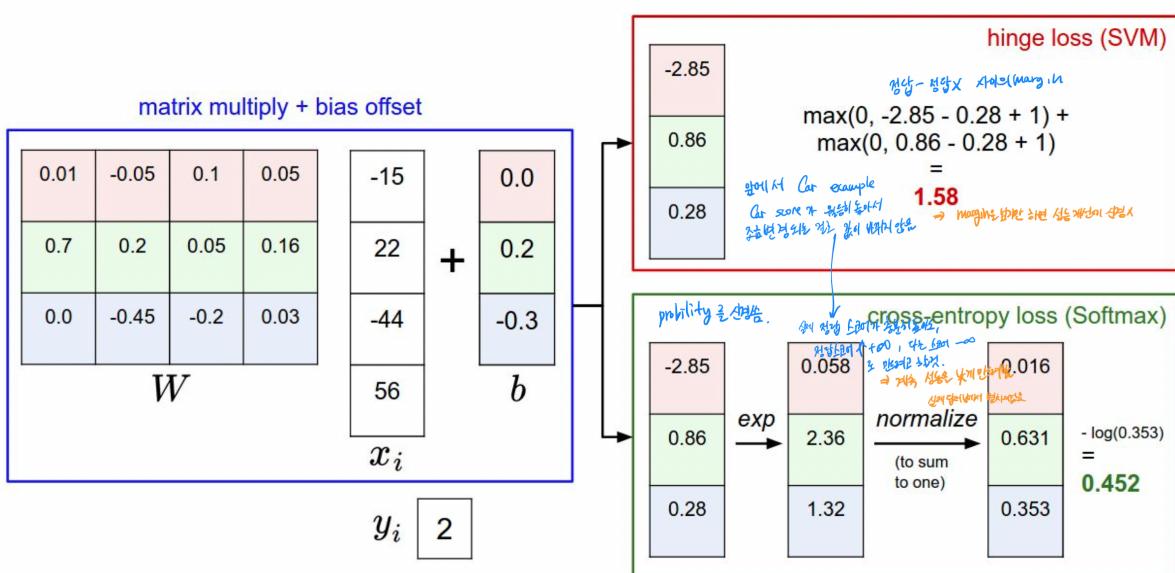
0.00

unnormalized log probabilities

probabilities

- What is the min/max possible loss L_i ?
 - $\min = 0$
 - $\max = \text{infinite}$
 - 두 값은 이론적인 값이고 실질적으로 도달 불가능하다.
 - 정답 클래스에 대한 로그 확률 : $-\log(1) = 0$
 - 정답 스코어가 매우 매우 높아야한다. (+infinite)
 - Usually at initialization W is small so all $s \sim 0$. What is the loss?
 - $-\log(1/c) = \log C$
 - 이는 디버깅할 때 좋다.

SVM and Softmax



SVM은 margin을 넘기만 하면 더이상의 성능 개선에는 신경쓰지 않음.

softmax는 계속 성능을 낮게 만들려고 한다.

Summary

Optimization

best W를 찾는 방법

Random search

가장 단순하고 나쁜 아이디어

```
# 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 9.044634, 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)
```

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

15.5% accuracy! not bad!
(SOTA is ~95%)

Follow the slope

slope = derivative of function

gradient는 편도함수의 벡터로, 선형 1st order approximation으로 구할 수 있다. gradient의 방향은 곧 함수에서 가장 많이 올라가는 방향을 가리키므로 negative gradient인 가장 내려가는 방향을 사용한다. 특정 방향에서 얼마나 가파른지는 해당 방향의 유닛벡터와 gradient 벡터를 내적한다.

Finite Difference Method

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	<p style="color: blue;">[-2.5, ?, ?, FDM로 정수 차원 gradient 계산하기 $(1.25322 - 1.25347)/0.0001$ $= -2.5$</p> <div style="border: 1px solid black; padding: 5px; width: fit-content;"> $\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$ </div> <p style="color: blue;">?, ?,...]</p>

first Dim을 초기화하면 loss도 대체로 증가함.

gradient dW를 모두 계산해서 평균치를 계산한다. 해당 방법은 매우 느리므로 실제로 사용하지 않는다.

Analytic gradient

FDM안 쓰고 calculus 사용한다.

항상 analytic gradient를 사용하지만 FDM으로 gradient check를 할 수 있다.

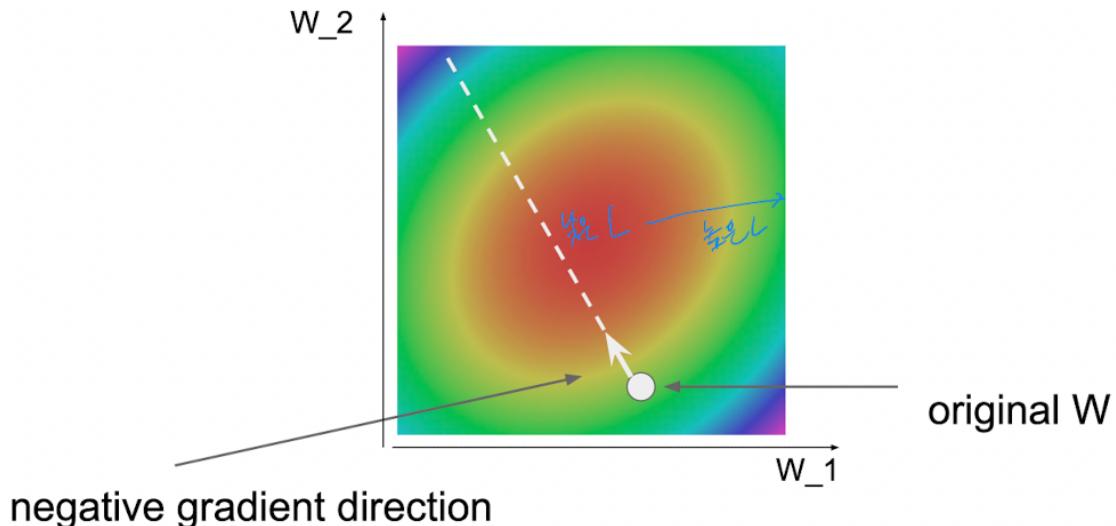
Gradient Descent

1. initialize W in random value
2. loss, gradient 계산
3. gradient 반대방향으로 update = decrease 방향

```
# Vanilla Gradient Descent

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

step_size : hyperparameter = learning rate



Stochastic Gradient Descent (SGD)

gradient를 한 번 계산하려면 전체 트레이닝 셋을 한 바퀴 돌아야한다. N이 커질 수록 매우 느려진다. 전체 G와 L을 구하기 보다 작은 트레이닝 샘플(minibatch, 2^n 의 개수)를 떼어서 계산하는 것이 좋다.

```
# 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
```

기존 이미지 추출방식

이미지 자체를 입력으로 사용하지 않고 2개의 단계가 있었다. feature를 뽑고 concat해서 이어준 뒤, linear classifier에 적용한다.

ex) color histogram (전체적으로 어떤 색인지 알 수 있다.)

Histogram of Oriented Gradients : 8x8 픽셀로 전체 이미지를 자른 후, 방향을 만들어 edge의 의존도를 추출한다.

Bag of words : 이미지의 작은 지점의 patch를 빈도 혹은 색의 vector로 설명한다. 이것을 사전화 하여 test image와 가장 유사한 특징 벡터를 찾는다.