Multi-class SVM loss

자매품으로 Softmax (multinomial logistic regression)도 많이 씀
— 특히 DL에서는 softmax를 더!

∘ multi-class SVM: binary-class SVM의 일반화된 형태
  ㄴ 두 개의 class만 다룸 ↔ CIFAR-10 has 10 classes.
  ㄴ positive/negative

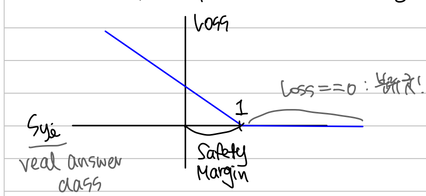
$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

= safety margin.

맞지 않는 카테고리에 대해

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

⇒ if) 정답 class의 score 점수 가장 높으면:
then) $\max(0, s_j - s_{y_i} + 1)$
Max(0, value)의 형태 – "hingle loss"



Q. Safety Margin?
스코어 간 상대적인 차이를 고려함.
사실 별로 상관없음 ~~

손실함수 L_i : 얼마나 구리게 예측? 정량화함.

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

최종 Loss "L": 각 N개 sample 들의 Loss 평균.

- W 탐색해가며 loss를 minimize하는 W 찾게 될것임.

multi-class SVM Loss
이진 SVM의 일반화된 형태
SVM (Support vector machine)

Multiclass SVM loss: Questions

Q. What happens to loss if car scores change a bit?
SVM loss는 "다른 스코어들과의 차이" 고려함.
일단 그 예제에서 Car Score가 상대적으로 엄청 크기 때문에 "조금" 바꿔어도 서로 간 Margin 유지
→ Loss change X

Q. What is the min/max possible loss?
- minimum: 0
  if every class, answer class score is very big
- maximum: ∞
  consider the loss function is hingle loss shape.
  if answer class score : 엄청 낮은 Neg Value.

Q. At initialization W is small so all $s \approx 0$. what is the loss?
: N(class) − 1.
all $s \approx 0$, and our margin is 1. Since, we got the loss: $C-1$
⇒ useful debugging strategy:
Loss가 어떻게 될지 짐작 가능. (if 트레이닝 첫 시작 시 Loss ≠ C−1, bug)

Q. what if the sum was over all classes? (including $j = y_i$)
정답인 class까지 다 더하면: Loss += 1
'원래' $j = y_i$인 class (정답 class): Loss == 0. ⇒ Loss가 0이면 잃는 것이 없다고 해석 가능하기 때문

Q. What if we used mean instead of sum?
그저 rescaling.
상대값으로 비교하므로 의미 없음

Q. What if we used $\sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$ — squared hinge loss.
(kind of tradeoffs between good and badness in kind of nonlinear way)
→ different. 본래 함수와 다른 양식

목적에 따라, 상황에 맞게 선택 {
hinge loss: squared 보다 안 좋은 것이 덜 안 좋은
squared hinge loss: 엄청 안 좋은 것 → 더 ×100 안 좋은 것
}

Example Code

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

numpy 이용해 손실함수 구현 ⇒

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

Question about loss function

$f(x, W) = Wx$

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

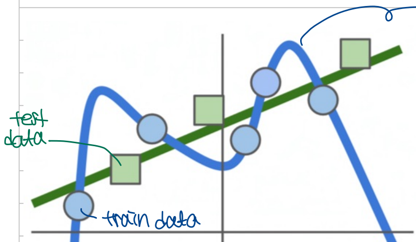
Q. Suppose that we found a W such that L=0. Is this W unique?

→ 다른 W도 존재.

ex) 2W is also has L=0.

If there exists W & 2W → 스코어 차이의 margin 또한 2배 차이 O

margin의 이미 > 1 : 2배를 해도 1. 따라서 loss는 똑같이 0이 맞다.



classifier: train data에 맞는 곡선 만듦 → performance ↓

new data 들어오면 완전히 틀림

우리는 training data가 아닌 test data에 적합한 model 찾는 중임.

→ training data의 loss에만 신경 ↓

test data에서의 performance 고려!! ∴ Regularization

Loss function now has:
- Data Loss
- Regularization Loss
+ Lamda (Hyperparameter)

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

① Data loss term: model predictions should match training data

손실 함수에 항 추가!

→ ② Regularization: model should be "simple", so it works on test data.

"Simple": 해결 문제, 상황에 따라 상이 - Occam's Razor: "Among competing hypothesis, the simple is the best"

일반적으로 더 단순한 것을 선호해야 함

→ 단순한 것이 미래 현상 설명 가능성 ↑

∴ Regularization penalty

Regularization → R
- 1) 모델 더 복잡해지는 것 방지
- 2) Soft penalty 부여

Regularization λ = regularization strength (hyperparameter)

Weight W에 대한 Euclidean norm (squared Norm) (or.. 1/2 squared Norm)

→ W에 대한 penalty 부여함.

(L1 Regularization은 L1으로 W에게 penalty 부여)

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

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

→ 딥러닝, 기계학습.. 많이 쓰임.

∴ Regulation: training set의 overfitting 방지하기 위해 penalty 부여하는 방법!.

TITLE DATE

36/85

L2 Regularization (Weight Decay)

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

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

(If you are a Bayesian: L2 regularization also corresponds MAP inference using a Gaussian prior on W)

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

Linear classifier의 관점에서, w1와 w2는 같은 하지만 L2 Regression은 norm이 더 작은 w2 선호

Linear classification에서 w: "얼마나 x가 output class와 닮았는지"

L2: 모든 x의 요소가 골고루 영향 주길 바랄 때 사용. (w2 선호)

L1: ↔ 정반대. "복잡도"를 다르게 정의함 (w1 선호)

가중치 w에, 0의 개수에 따라 정의. (0 ↑: 복잡)

Spare solution 선호: w의 요소 중 대부분이 0가 될것임.

TITLE Softmax (Multinomial logistic Regression) DATE

SVM: 스코어 '자체'에 대해 고려 X. 상대적인 차이값만 고려함

↔ Softmax: 스코어 '자체'에 대해 고려함. What those scores means?

→ 클래스별 확률 분포 계산 O

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

Probabilities

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

0-1 → sum: 1

Softmax function (positive value)

Probability Distribution

cat	**3.2**
car	5.1
frog	-1.7

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

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

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

∵ probability는 1에 가까울수록 정답에 가까운 것 (good)

↔ loss: 정답과 먼 것 (bad)

→ 음수를 취해 줌 minus log of the probabilities.

Softmax Classifier (Multinomial Logistic Regression)

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

unnormalized probabilities

	unnormalized log probabilities	loss function exp →	unnormalized probabilities	to sum=1 normalize → 합이 1이 되도록 정규화	probabilities
cat	**3.2**		**24.5**		**0.13** → L_i = -log(0.13) = **0.89**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

Q. What is the min/max possible loss Li?

- min: 0 (log(1)=0) answer class.
- max: ∞ (log(0+-) = ∞)

answer class: 1
~ : 0

loss 구할 때 minus log 취해 주는 것!

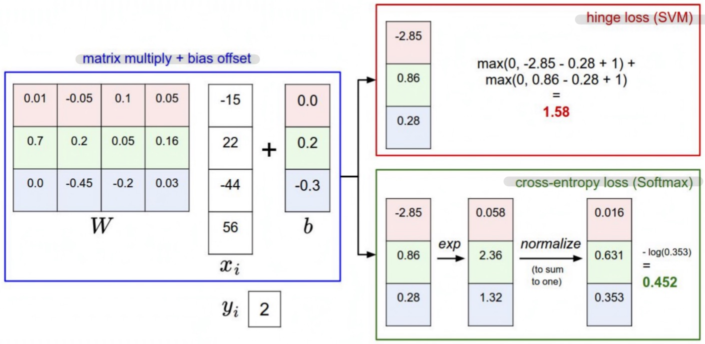
Q. Usually at initialization W is small so as s≈0. What is the loss? $-\log(1/C) = \log C$

→ log C가 아니면 뭔가 잘못된것..

MADE BY JANE

TITLE DATE

"How Badness the W is?"
를 측정하는 방법이 모두 다름.



Softmax vs SVM

very high score / low score → always consider datapoint (to get better)

consider only thing: "margin" datapoint → just only use to get value.

Recap

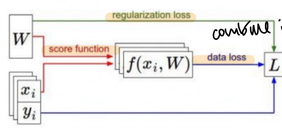
How do we find the best W? → Optimization!

- 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**: How quantify bad the W is.

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

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

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



combine it together

TITLE Optimization DATE

Setting W. → "iterative." improve it every time

Stategy #1. Random Search

Really Bad Algorithm !!

Stategy #2. Follow the Slope

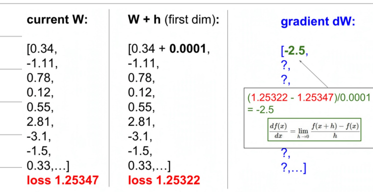
where is the slop of this? → go ahead little bit..

Get well! General Stategy.

1-dim func: $\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$

multi-dim func: gradient is the vector of (partial derivatives) along each dimension.

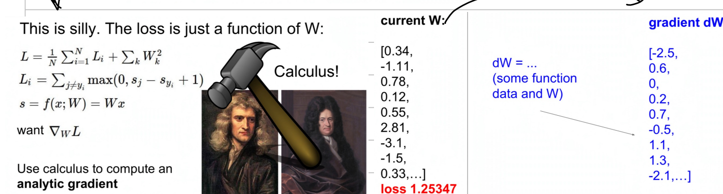
tell us what is the function f that we move in backward direction.



Super slow !!

use math

go directly (one step)



way more efficient. fast

(In practice: Always use analytic gradient, but check implementation with numercial gredient.)

→ "griedient check"

very useful debugging stategy.

super slow.. but super useful for debugging..

TITLE # Gradient Descent

DATE

code

```
# Vanilla Gradient Descent

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

hyper parameter

(learning rate, step size : 사전에 확인하는 parameter.)

Stochastic Gradient Descent (SGD)

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

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

Full sum expensive when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

many data → so slow
rather than computing entire set,
we minibatch
small → estimate of the small sum.

update gradient with this estimate!

code

```
# 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
```

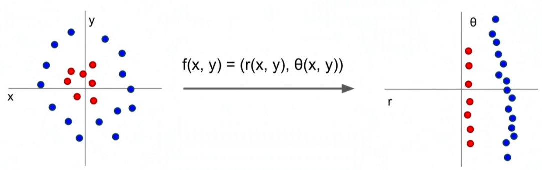
Image Features

→ linear classifier 별로 X ∵ Multi-modality

1) calculate feature
→ vector → to linear classifier input.

Motivation → Feature transform

Image Features: Motivation



Cannot separate red and blue points with linear classifier

After applying feature transform, points can be separated by linear classifier

NN이 뜨기 전, HOG (Local origentation edges 측정) 이용