Лекция 3:

Функции потерь и оптимизация

Распознавание образов: основная задача машинного зрения

This image by Nikita is licensed under CC-BY 2.0

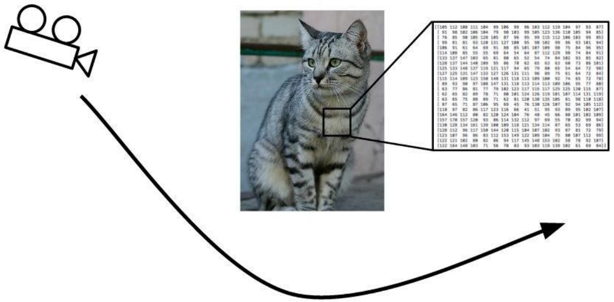
Задача: определить класс изображения:
{dog, cat, truck, plane, ...}

→

cat
dog
bird
deer
truck

Проблемы распознавания изображений

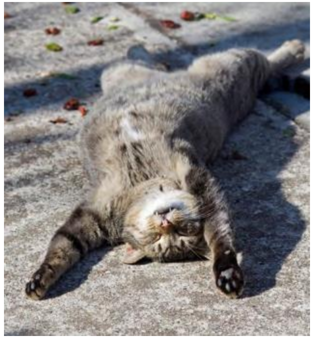
Viewpoint



Illumination

This image is CC0 1.0 public domain

Deformation



This image by Umberto Salvagnin
is licensed under CC-BY 2.0

Occlusion

This image by jonsson is licensed
under CC-BY 2.0

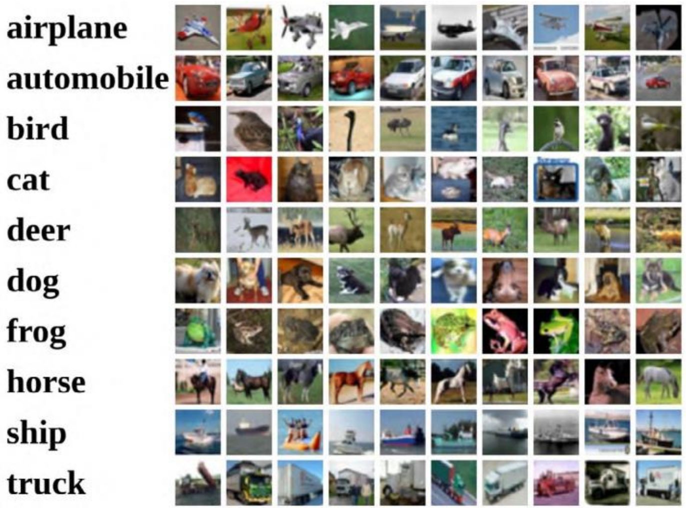
Clutter

This image is CC0 1.0 public domain

Intraclass Variation

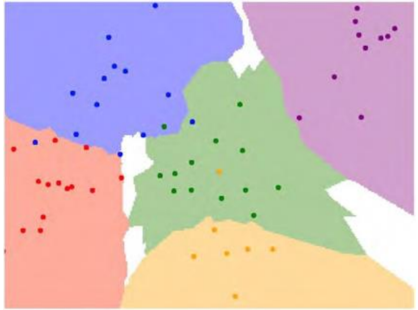
This image is CC0 1.0 public domain

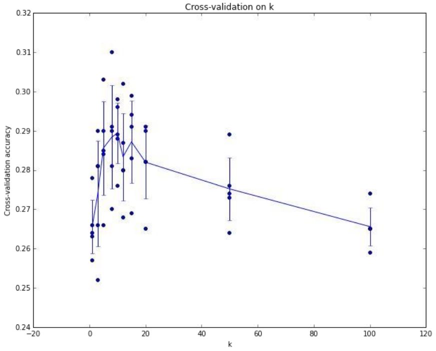
Возможный вариант: kNN



1-NN classifier

5-NN classifier

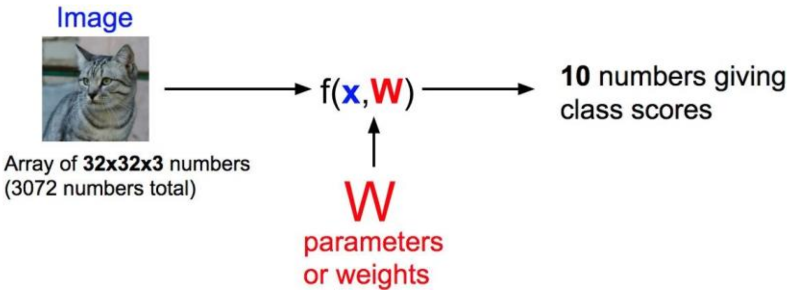




train	test

train	validation	test

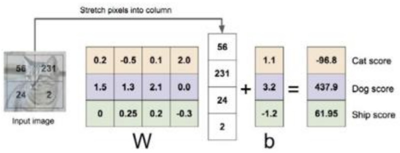
Возможный вариант: Линейная классификация



$$f(x,W) = Wx + b$$

Algebraic Viewpoint

$f(x,W) = Wx$

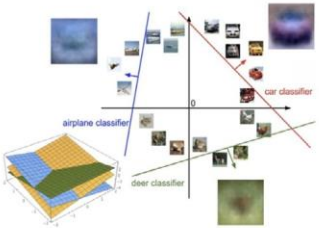


Visual Viewpoint

One template
per class

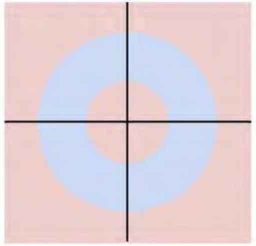
Geometric Viewpoint

Hyperplanes
cutting up space



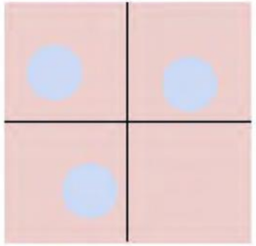
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else

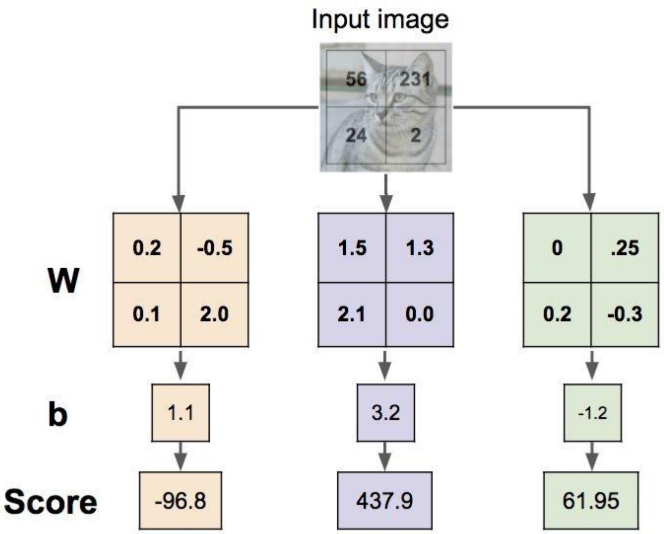


Class 1:
Three modes

Class 2:
Everything else



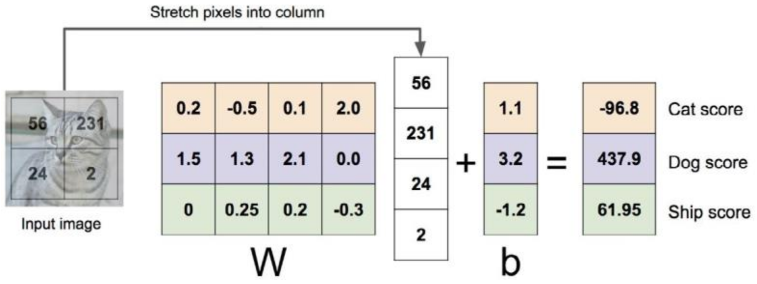
Интерпретация линейного классификатора



Вырожденный пример, 4 пиксела, 3 класса (cat/dog/ship)

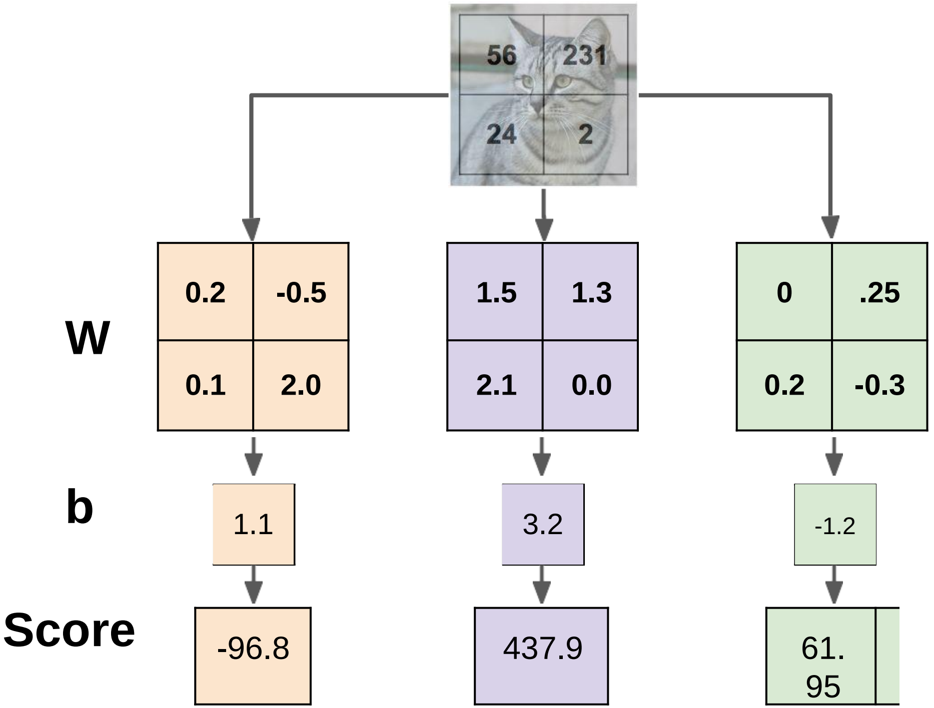
Algebraic Viewpoint

$$f(x,W) = Wx$$

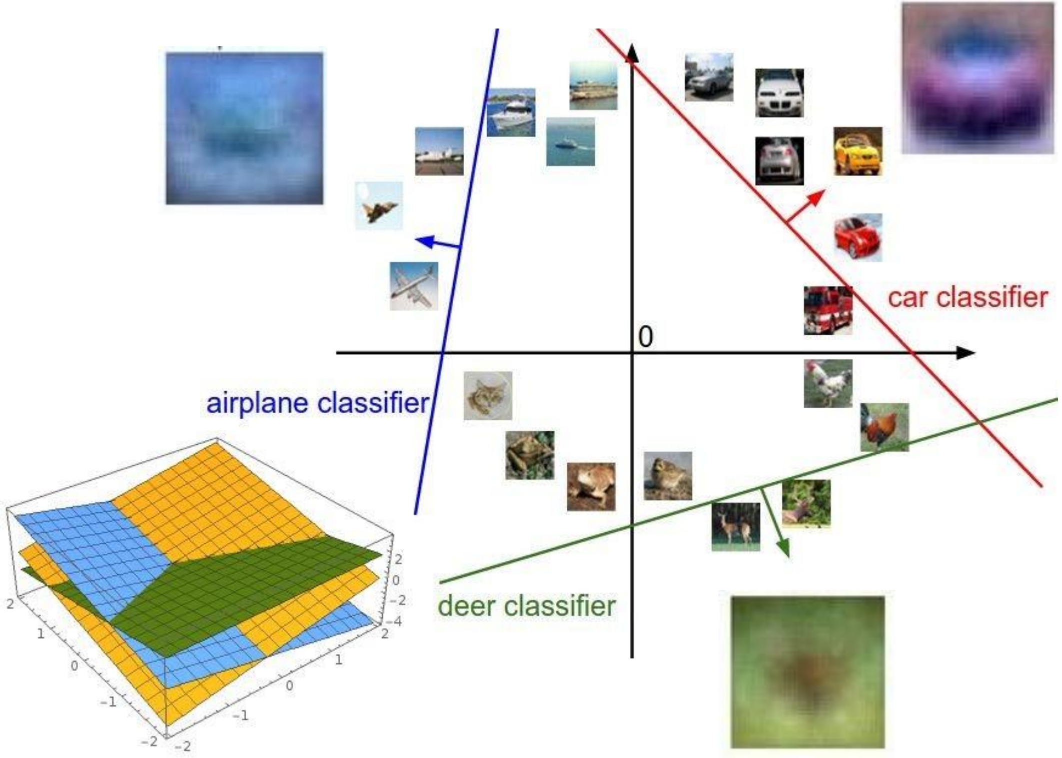


Visual Viewpoint

Input image



Геометрическая интерпретация



$$f(x,W) = Wx + b$$

Array of **32x32x3** numbers
(3072 numbers total)

Plot created using Wolfram Cloud

Линейный классификатор



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

Score matrix = Матрица оценок

Что нам нужно:

1. Функция потерь **loss function** – определяет насколько мы не правы.

2. Процедура оптимизации функции потерь. **(optimization)**

Дано: 3 примера, 3 класса. С параметрами W и оценками $f(x, W) = Wx$



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

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 average of loss over examples:

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

Дано: 3 примера, 3 класса. С параметрами W и оценками $f(x, W) = Wx$



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

Функция потерь (loss) задается как:

Для обучающей выборки

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

Где x_i картинка

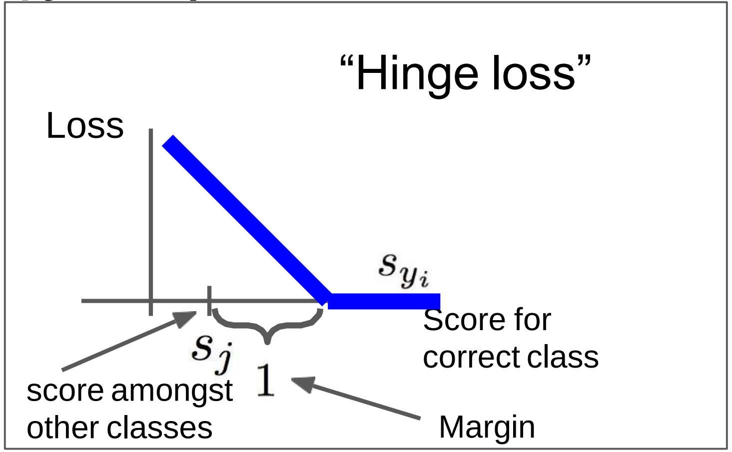
y_i номер класса

Функция потерь по выборке задается как усреднение потерь на каждом примере:

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

Дано: 3 примера, 3 класса. С параметрами W и оценками $f(x, W) = Wx$

Multiclass SVM loss (шарнирная функция):





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

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

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

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

Дано: 3 примера, 3 класса. С параметрами W и оценками $f(x,W) = Wx$

Multiclass SVM loss:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

SVM loss:

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

= 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

Дано: 3 примера, 3 класса. С параметрами W и оценками $f(x, W) = Wx$

Multiclass SVM loss:

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	

SVM loss:

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

= 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

Дано: 3 примера, 3 класса. С параметрами W и оценками $f(x, W) = Wx$

Multiclass SVM loss:

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

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

= max(0, 2.2 - (-3.1) + 1)
  +max(0, 2.5 - (-3.1) + 1)
= max(0, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9

Дано: 3 примера, 3 класса. С параметрами W и оценками $f(x, W) = Wx$

Multiclass SVM loss:

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

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

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**

В итоге, минимум достигнут во втором столбце матрицы оценок $f(x, W) = Wx$

Multiclass SVM loss:

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

cat	1.3
car	**4.9**
frog	2.0
Losses:	0

Q1: Что будет с функцией потерь, если оценка для машинки уменьшить на 0.5, а на 1?

Q2: min/max значения для SVM loss L_i?

Q3: При инициализации $W_{ij} \approx 0$ и $s_j \approx 0$. Чему равно L_i, для N примеров C классов?

Матрица оценок:

$$f(x, W) = Wx$$



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

the SVM loss has the form:

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

Q4: а что будет, если включать все классы?

Матрица оценок:

$$f(x, W) = Wx$$

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

the SVM loss has the form:

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

Q5: Что будет если брать среднее вместо суммы?

Матрица оценок:

$f(x, W) = Wx$ 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	0	12.9

the SVM loss has the form:

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

Q6: а квадрат?

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

Multiclass SVM Loss: Код

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

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

\# First calculate scores
\# Then calculate the margins $s_j - s_{yi} + 1$
\# only sum j is not y_i, so when $j = y_i$, set to zero.
\# sum across all j

$$f(x, W) = Wx$$

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

Q7. Пусть W такое что L = 0.
W единственное?

$$f(x, W) = Wx$$

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

Q7. Пусть W такое что L = 0.
W единственное?

Нет для 2W L=0!
Что будем делать?

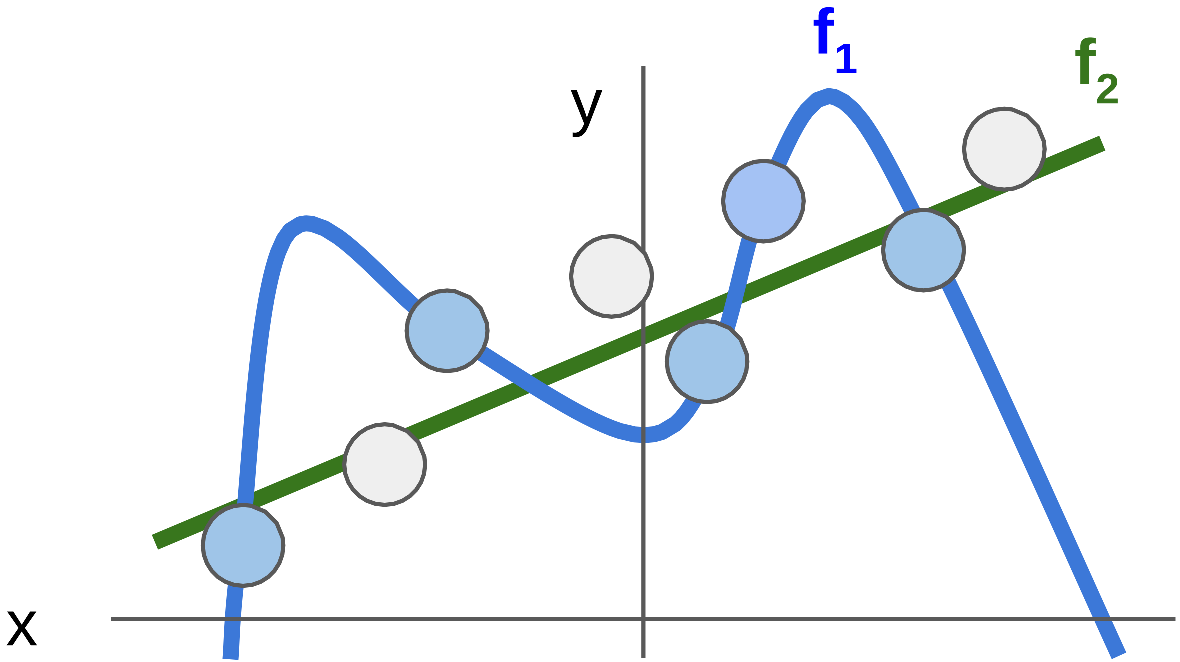
Регуляризация(Regularization)

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

Data loss: ошибка классификации

Regularization: ограничение на веса модели

Регуляризация: чем проще тем лучше!



Уберем переобучение! – Бритва Оккама

Регуляризация

λ = гиперпараметр силы регуляризации

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

Data loss: ошибка классификации

Regularization: ограничение на веса модели

Примеры:

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

Более сложные:

Dropout

Batch normalization

Stochastic depth, fractional pooling

Регуляризация(Regularization)

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

Data loss: ошибка классификации

Regularization: ограничение на веса модели

Зачем это делать?

- Указать доп. требования к весам
- Упростить модель
- Сделать задачу оптимизации более выпуклой

Пример: требования к весам

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

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

L2 регуляризация

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

Что лучше для L2 регуляризации w1 или w2?

Пример: требования к весам

$$x = [1, 1, 1, 1]$$

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

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

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

L2 регуляризация

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

Что лучше для L2 регуляризации w1 или w2?

L2 регуляризация даст более гладкое распределение весов.

А как на счет L1?

Softmax классификатор

Softmax (Мультиномиальная логистическая регрессия)

Сделаем из оценок **вероятности:**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Функция Softmax

Вероятности >= 0

Вероятности нормированы к 1

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

	ненормированные логиты		Ненормированные вероятности		вероятности
cat	3.2	exp →	24.5	normalize →	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

→ $L_i = -\log(0.13)$
$= \mathbf{2.04}$

И это оценка максимального правдоподобия!

Softmax (Мультиномиальная логистическая регрессия)

Сделаем из оценок **вероятности:**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Функция Softmax

Вероятности >= 0

Вероятности нормированы к 1

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

	ненормированные логиты		Ненормированные вероятности		вероятности		Correct probs
cat	**3.2**	exp →	**24.5**	normalize →	**0.13**	→ сравним ←	**1.00**
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Дивергенция Кульбака-Лейблера

$$D_{KL}(P\|Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

Softmax (Мультиномиальная логистическая регрессия)

Сделаем из оценок **вероятности:**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Функция Softmax

Вероятности >= 0

Вероятности нормированы к 1

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

		exp		normalize			
cat	**3.2**		**24.5**		**0.13**	→ сравним ←	**1.00**
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00
	ненормированные логиты		Ненормированные вероятности		вероятности		Correct probs

Кросс Энтропия

$$H(P, Q) = H(p) + D_{KL}(P \| Q)$$

Softmax (Мультиномиальная логистическая регрессия)

Сделаем из оценок **вероятности:**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Функция Softmax

cat	**3.2**
car	5.1
frog	-1.7

Максимизируем вероятность верного класса

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

Все вместе:

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

Softmax (Мультиномиальная логистическая регрессия)

Сделаем из оценок **вероятности:**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Функция Softmax

cat	**3.2**
car	5.1
frog	-1.7

Максимизируем вероятность верного класса

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

Все вместе:

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

Q1: Какие min/max значения для softmax L_i?

Q2: Инициализируем s_j примерно равными;
Какое значение примет L_i, для C классов?

Softmax (Мультиномиальная логистическая регрессия)

Сделаем из оценок **вероятности:**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Функция Softmax

cat	**3.2**
car	5.1
frog	-1.7

Максимизируем вероятность верного класса

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

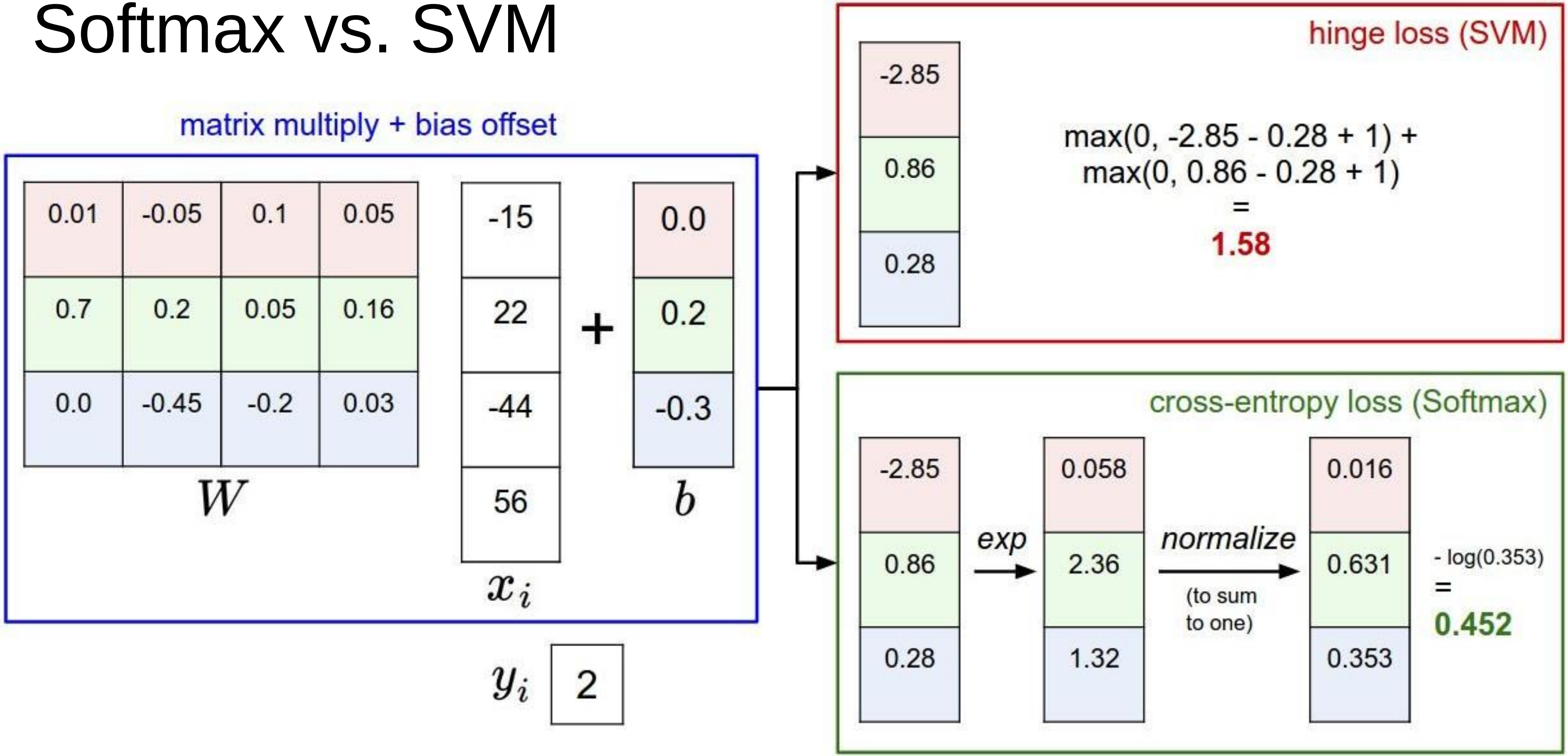
Все вместе:

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

Q1: Какие min/max значения для softmax L_i?

Q2: Инициализируем s_j примерно равными;
Какое значение примет L_i, для C классов?

Softmax vs. SVM



Softmax vs. SVM

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

оценки:
[10, -2, 3]
[10, 9, 9]
[10, -100, -100]
and $\boxed{y_i = 0}$

Q1: Посчитайте функцию потерь для SVM и Softmax?

Q2: а если оценку для нулевого класса увеличить с 10 до 20?

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**:

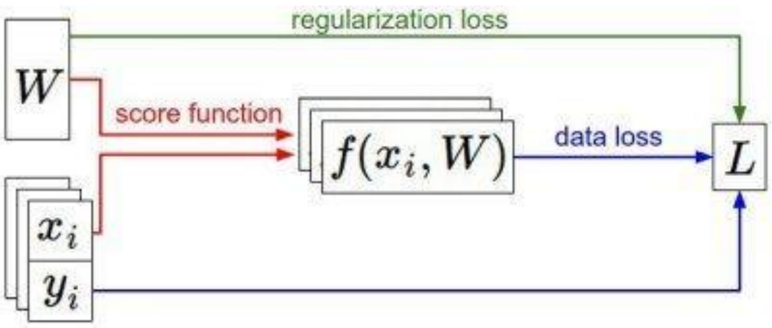
Softmax

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

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



Оптимизация

Walking man image is CC0 1.0 public domain

Strategy #1: Плохая идея: **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
# ... (trunctated: continues for 1000 lines)
```

Проверим на тесте...

```
# 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 ~99.3%)

Strategy #2: **Вниз по склону**

Скалярный случай - производная:

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

В векторном случае – градиент, нам надо идти в направлении антиградиента

Как его брать?

Численный градиент(плохая идея)

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (first dim)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25322

gradient dW:

[**-2.5**,
?,
?,
?,
?,…]

(1.25322 - 1.25347)/0.0001
= -2.5

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

current W:	**W + h** (second dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + **0.0001**,	**0.6**,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25353**	

(1.25353 - 1.25347)/0.0001
= 0.6

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

current W:	**W + h** (third dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	**0**,
0.12,	0.12,	?,
0.55,	0.55,	?
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25347**	

(1.25347 - 1.25347)/0.0001
= 0

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

Все просто, возьмем градиент по W:

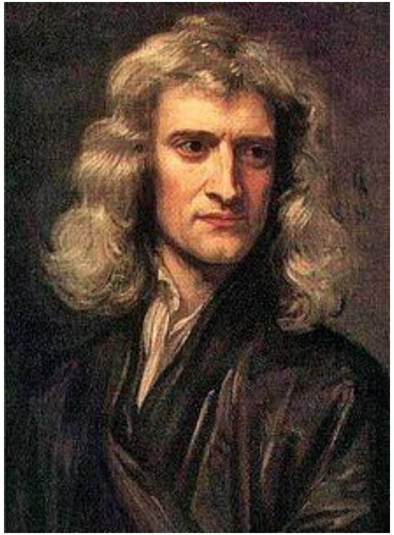
$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

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

$$s = f(x; W) = Wx$$

want $\nabla_W L$

Use calculus to compute an
analytic gradient



This image is in the public domain

This image is in the public domain

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

dW = ...
(некая функция от данных и W)

gradient dW:

[-2.5,
0.6,
0,
0.2,
0.7,
-0.5,
1.1,
1.3,
-2.1,…]

В итоге:

- Численный градиент медленный, приближенный, но простой в реализации

- Аналитический градиент: точный, быстрый, можно ошибиться!

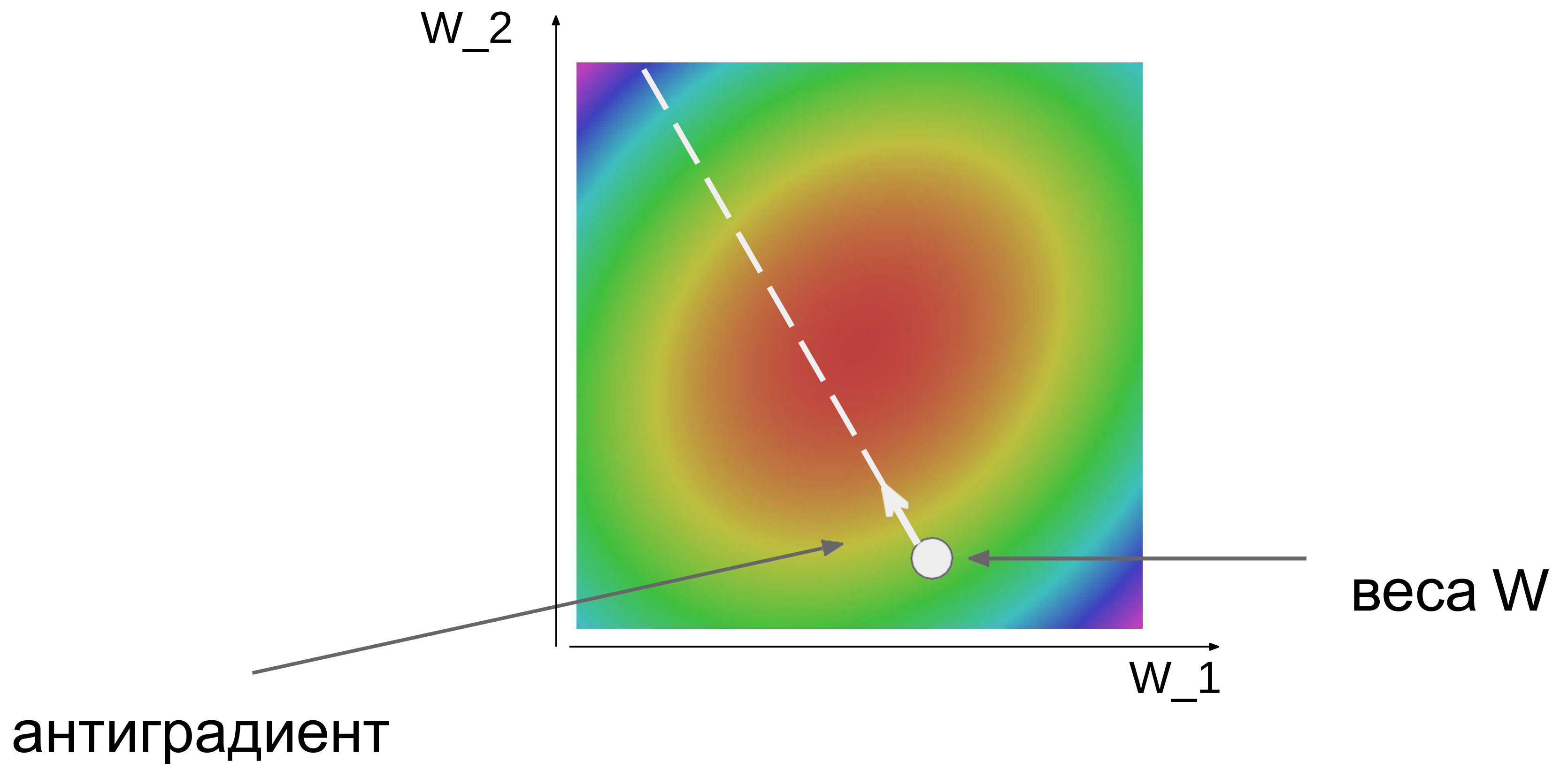
=>

<u>На практике:</u> Используем аналитический градиент но сверяем с численным - **gradient check.**

Gradient Descent / Градиентный спуск

```
# Vanilla Gradient Descent

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



W_2
W_1
веса W
антиградиент

Stochastic Gradient Descent (SGD)

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

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

Полная сумма очень большая при большом объеме данных!

Используем частичныек суммы - **minibatch** по 32 / 64 / 128 строк

```
# 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
```

Дальше:

Вычислительные графы
нейронных сетей

Backpropagation