

Lecture 3:

Loss functions and

Optimization

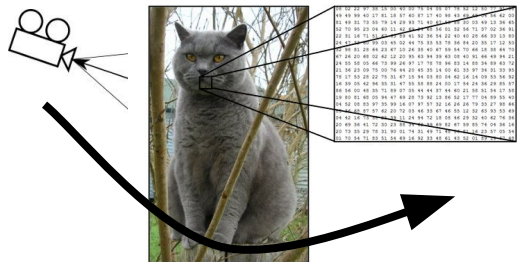
Administrative

A1 is due Jan 20 (Wednesday). ~9 days left

Warning: Jan 18 (Monday) is Holiday (no class/office hours)

Recall from last time... Challenges in Visual Recognition

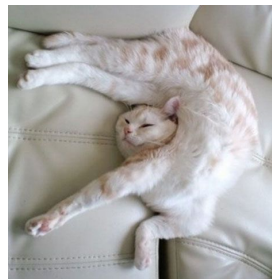
Camera pose



Illumination



Deformation



Occlusion



Background clutter



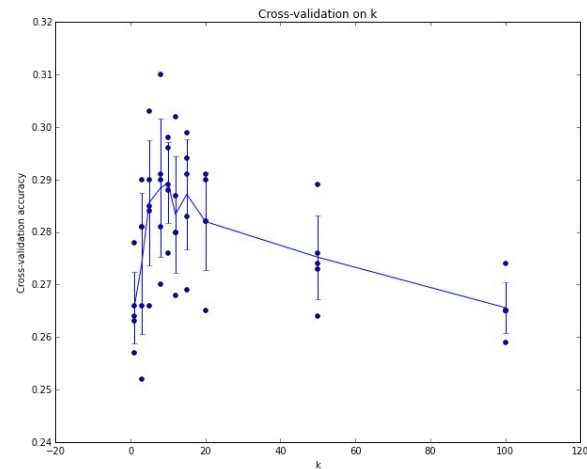
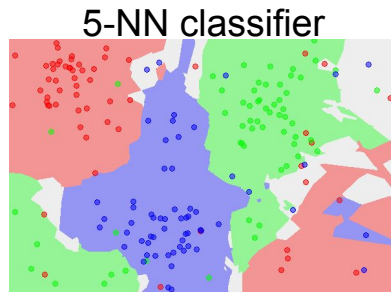
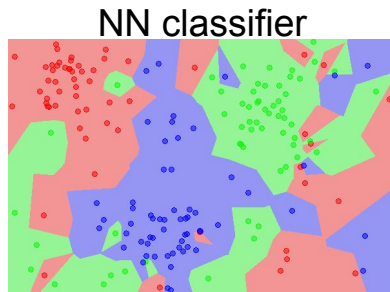
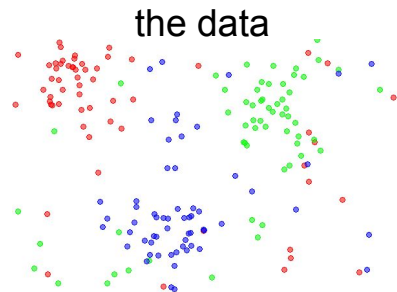
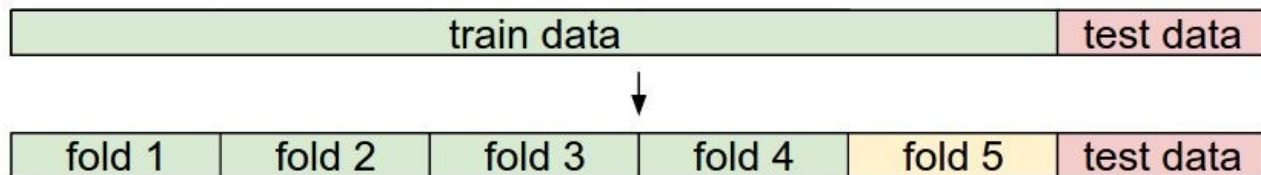
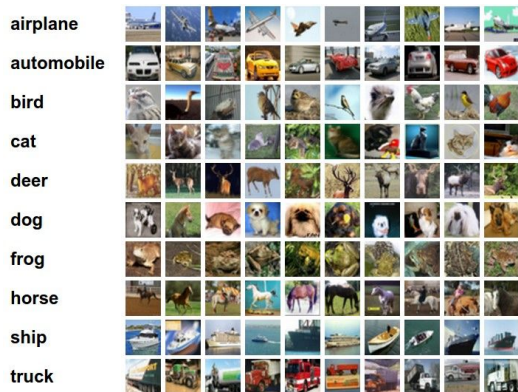
Intraclass variation



- This is a very difficult problem
- We must be robust to all these variations.
- State of the art can solve these problems

Recall from last time... data-driven approach, kNN

We can't explicitly hard code these classifiers so we have to train them from data.



Recall from last time... Linear classifier

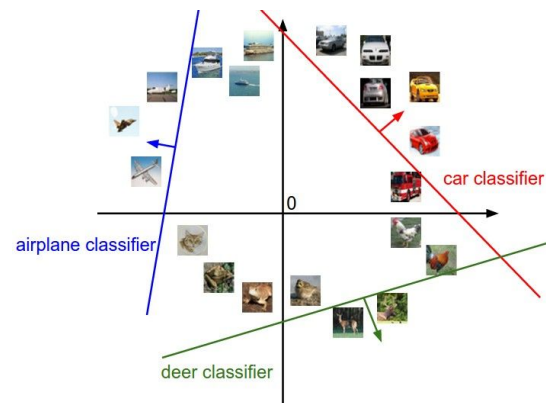
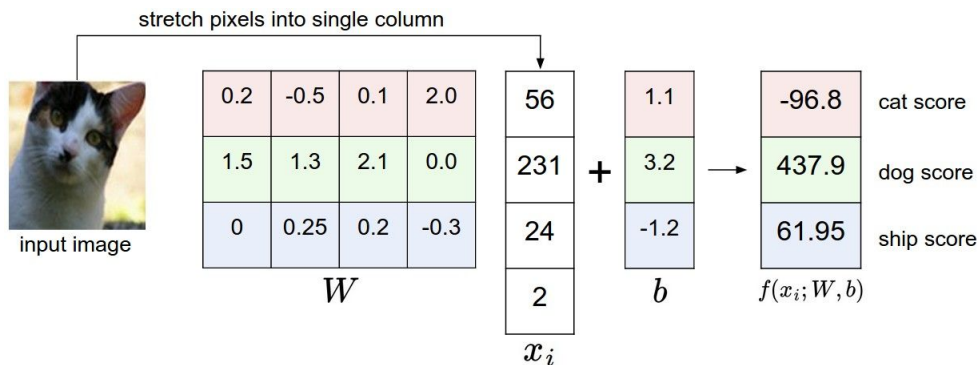


[32x32x3]

array of numbers 0...1
(3072 numbers total)

image parameters
 $f(\mathbf{x}, \mathbf{W})$

10 numbers, indicating
class scores



Recall from last time... Going forward: Loss function/Optimization



| | | | |
|------------|------------|-------------|--------------|
| 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 |



TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.
2. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

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 |



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 |

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:

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



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:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \max(0, 5.1 - 3.2 + 1) \\ &\quad + \max(0, -1.7 - 3.2 + 1) \\ &= \max(0, 2.9) + \max(0, -3.9) \\ &= 2.9 + 0 \\ &= 2.9 \end{aligned}$$

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 | 0 | |

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:

$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\
 &= \max(0, 1.3 - 4.9 + 1) \\
 &\quad + \max(0, 2.0 - 4.9 + 1) \\
 &= \max(0, -2.6) + \max(0, -1.9) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$

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 | 0 | 10.9 |

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:

$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\
 &= \max(0, 2.2 - (-3.1) + 1) \\
 &\quad + \max(0, 2.5 - (-3.1) + 1) \\
 &= \max(0, 5.3) + \max(0, 5.6) \\
 &= 5.3 + 5.6 \\
 &= 10.9
 \end{aligned}$$

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 | 0 | 10.9 |

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:

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

and the full training loss is the mean
 over all examples in the training data:

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

$$L = (2.9 + 0 + 10.9)/3 \\ = 4.6$$

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 | 0 | 10.9 |

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:

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

Q: what if the sum
 was instead over all
 classes?
 (including $j = y_i$)



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 | 0 | 10.9 |

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:

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

Q2: what if we used a
 mean instead of a
 sum here?



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 | 0 | 10.9 |

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:

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

Q3: what if we used

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



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 | 0 | 10.9 |

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:

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

Q4: what is the
 min/max possible
 loss?



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 | 0 | 10.9 |

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i the image and
 where y_i the (integer) label,

and using the shorthand for the scores
 vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Q5: usually at
 initialization W are small
 numbers, so all $s \approx 0$.
 What is the loss?



Example numpy 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  
    loss_i = np.sum(margins)  
    return loss_i
```

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

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



There is a bug with the loss:

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

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



There is a bug with the loss:

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

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



E.g. Suppose that we found a W such that $L = 0$.
Is this W unique?



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 | 0 | |

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

Before:

$$\begin{aligned}
 &= \max(0, 1.3 - 4.9 + 1) \\
 &\quad + \max(0, 2.0 - 4.9 + 1) \\
 &= \max(0, -2.6) + \max(0, -1.9) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$

With W twice as large:

$$\begin{aligned}
 &= \max(0, 2.6 - 9.8 + 1) \\
 &\quad + \max(0, 4.0 - 9.8 + 1) \\
 &= \max(0, -6.2) + \max(0, -4.8) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$



Weight Regularization

λ = regularization strength
(hyperparameter)

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

In common use:

L2 regularization

L1 regularization

Elastic net (L1 + L2)

Max norm regularization (might see later)

Dropout (will see later)



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

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

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

L2 regularization: motivation

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

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

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

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

Softmax Classifier (Multinomial Logistic Regression)



| | |
|------|------------|
| cat | 3.2 |
| car | 5.1 |
| frog | -1.7 |

Softmax Classifier (Multinomial Logistic Regression)



scores = unnormalized log probabilities of the classes.

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

| | |
|------|------------|
| cat | 3.2 |
| car | 5.1 |
| frog | -1.7 |

Softmax Classifier (Multinomial Logistic Regression)



scores = unnormalized log probabilities of the classes.

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

| | |
|------|-------------|
| cat | 3.2 |
| car | 5.1 |
| frog | -1.7 |

Softmax Classifier (Multinomial Logistic Regression)



scores = unnormalized log probabilities of the classes.

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

| | |
|------|------|
| cat | 3.2 |
| car | 5.1 |
| frog | -1.7 |

Softmax function

Softmax Classifier (Multinomial Logistic Regression)



scores = unnormalized log probabilities of the classes.

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

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

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

| | |
|------|-------------|
| cat | 3.2 |
| car | 5.1 |
| frog | -1.7 |

Softmax Classifier (Multinomial Logistic Regression)



| | |
|------|-------------|
| cat | 3.2 |
| car | 5.1 |
| frog | -1.7 |

scores = unnormalized log probabilities of the classes.

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

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

Softmax Classifier (Multinomial Logistic Regression)



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

cat

3.2

car

5.1

frog

-1.7

unnormalized log probabilities

Softmax Classifier (Multinomial Logistic Regression)



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

unnormalized probabilities

cat
car
frog

3.2

5.1

-1.7

exp

24.5

164.0

0.18

unnormalized log probabilities

Softmax Classifier (Multinomial Logistic Regression)



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

unnormalized probabilities

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

unnormalized log probabilities

probabilities

Softmax Classifier (Multinomial Logistic Regression)



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

unnormalized probabilities

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

unnormalized log probabilities

probabilities

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

Softmax Classifier (Multinomial Logistic Regression)



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

unnormalized probabilities

Q: What is the min/max possible loss L_i ?

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

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

unnormalized log probabilities

probabilities

Softmax Classifier (Multinomial Logistic Regression)



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

unnormalized probabilities

Q5: usually at initialization W are small numbers, so all $s \approx 0$. What is the loss?

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

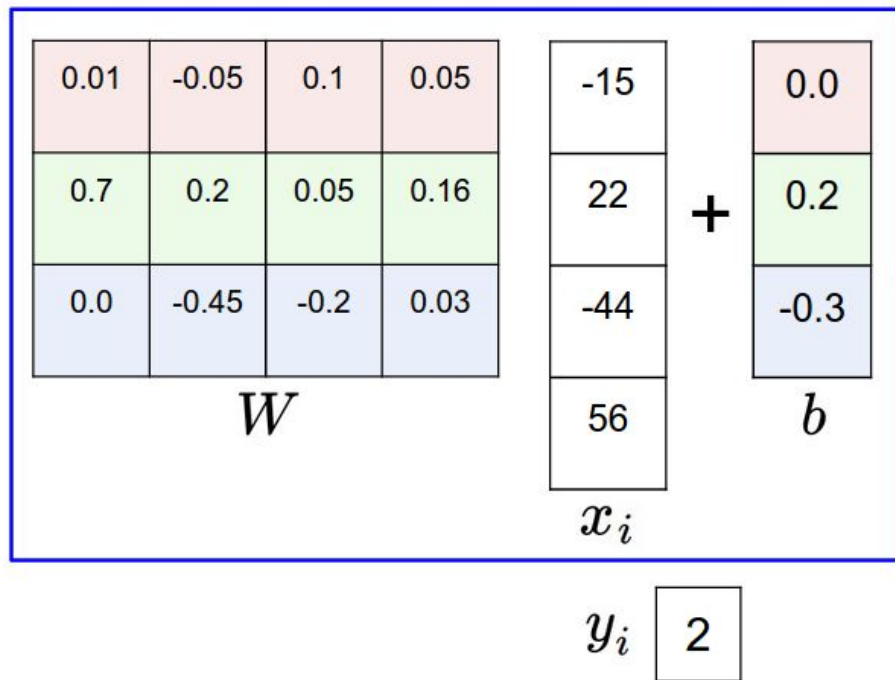
0.13
0.87
0.00

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

unnormalized log probabilities

probabilities

matrix multiply + bias offset



hinge loss (SVM)

| |
|-------|
| -2.85 |
| 0.86 |
| 0.28 |

$$\begin{aligned} &\max(0, -2.85 - 0.28 + 1) + \\ &\max(0, 0.86 - 0.28 + 1) \\ &= \\ &\mathbf{1.58} \end{aligned}$$

cross-entropy loss (Softmax)

| |
|-------|
| -2.85 |
| 0.86 |
| 0.28 |

$\xrightarrow{\text{exp}}$

| |
|-------|
| 0.058 |
| 2.36 |
| 1.32 |

$\xrightarrow{\text{normalize}}$
(to sum to one)

| |
|-------|
| 0.016 |
| 0.631 |
| 0.353 |

$$\begin{aligned} &-\log(0.353) \\ &= \\ &\mathbf{0.452} \end{aligned}$$

Softmax vs. SVM

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

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

Softmax vs. SVM

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

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

assume scores:

[10, -2, 3]

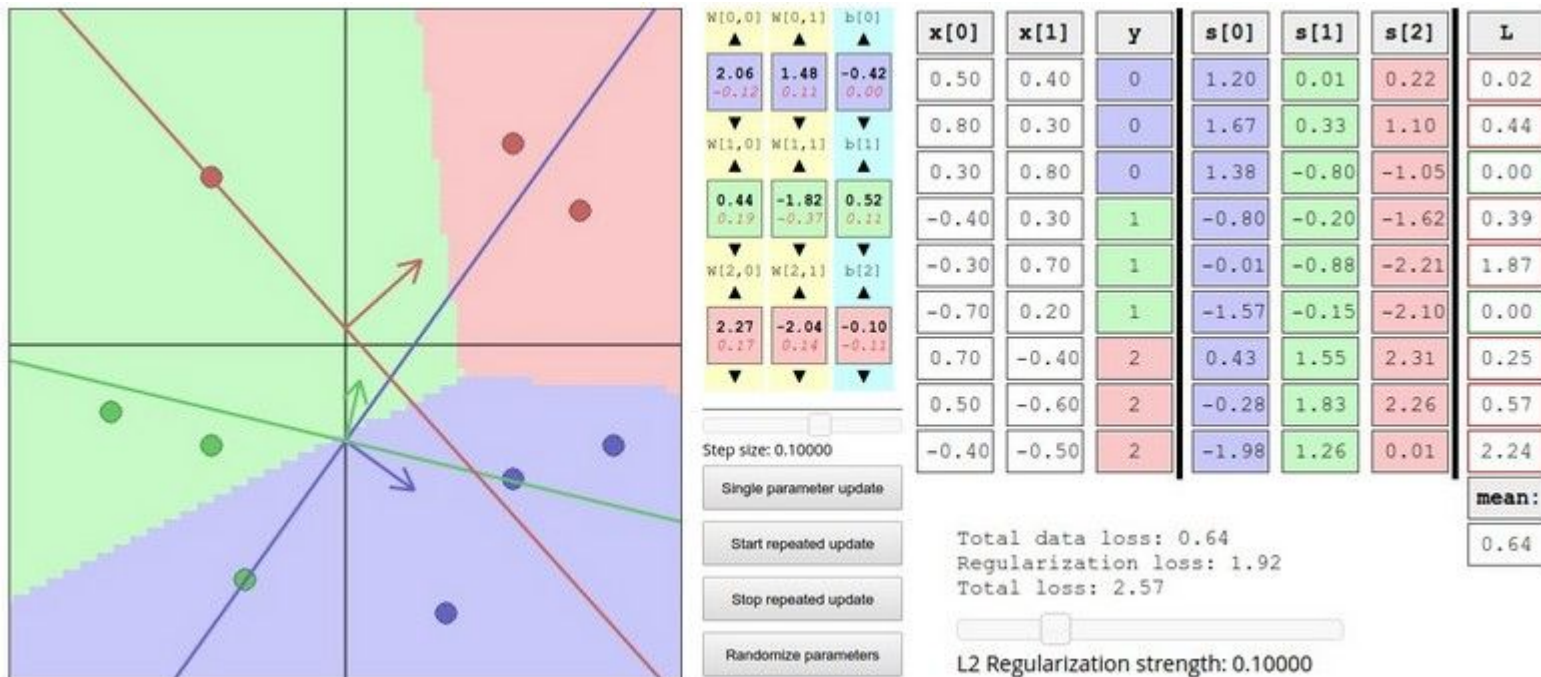
[10, 9, 9]

[10, -100, -100]

and $y_i = 0$

Q: Suppose I take a datapoint and I jiggle a bit (changing its score slightly). What happens to the loss in both cases?

Interactive Web Demo time....



<http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/>

Optimization

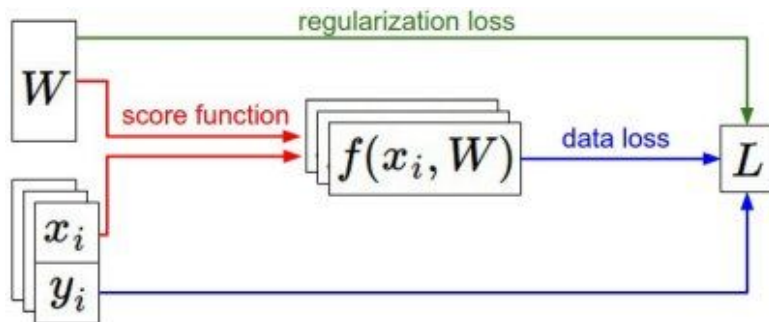
Recap

- We have some dataset of (x,y)
- We have a **score function**: $s = f(x; W) \stackrel{\text{e.g.}}{=} Wx$
- We have a **loss function**:

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right) \quad \text{Softmax}$$

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

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



Strategy #1: A first very bad idea solution: **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)

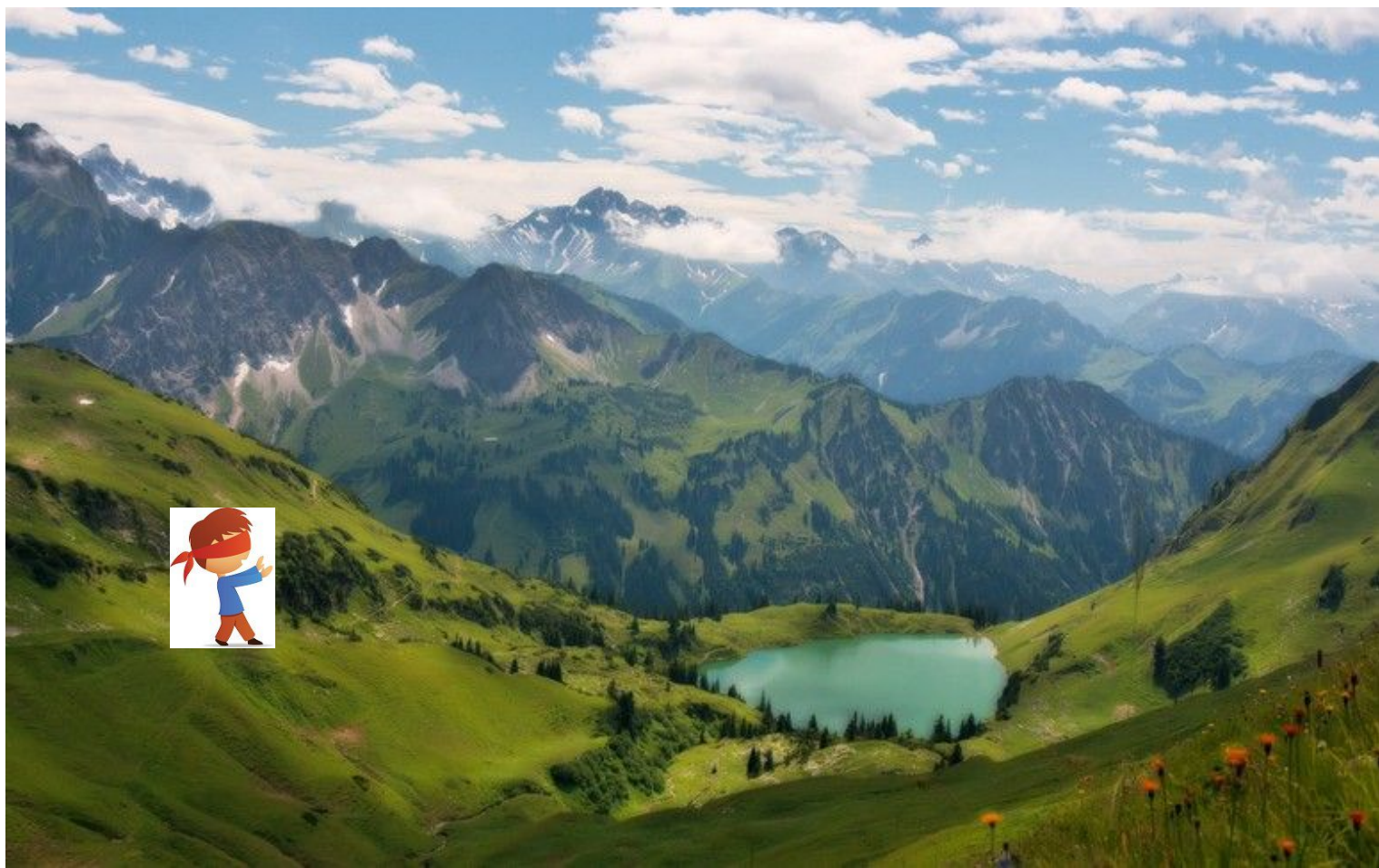
# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (truncated: continues for 1000 lines)
```

Lets see how well this works on the test set...

```
# 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%)





Strategy #2: **Follow the slope**

In 1-dimension, the derivative of a function:

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

In multiple dimensions, the **gradient** is the vector of (partial derivatives).

current W:

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

loss 1.25347

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,...]

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:

[?,
?,
?,
?,
?,
?,
?,
?,
?,...]

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,
?,
?,

$$(1.25322 - 1.25347)/0.0001 = -2.5$$

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

?,
?,...]

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 (second dim):

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

loss 1.25353

gradient dW:

[-2.5,
?,
?,
?,
?,
?,
?,
?,
?,...]

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 (second dim):

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

loss 1.25353

gradient **dW**:

[-2.5,
0.6,
?,
?,

$$(1.25353 - 1.25347)/0.0001 = 0.6$$

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

?,...]

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 (third dim):

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

loss 1.25347

gradient dW:

[-2.5,
0.6,
?,
?,
?,
?,
?,
?,
?,...]

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 (third dim):

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

loss 1.25347

gradient **dW**:

[-2.5,
0.6,
0,
?,
?

$$(1.25347 - 1.25347)/0.0001 = 0$$

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

?, ...]

Evaluation the gradient numerically

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

```
def eval_numerical_gradient(f, x):  
    """  
    a naive implementation of numerical gradient of f at x  
    - f should be a function that takes a single argument  
    - x is the point (numpy array) to evaluate the gradient at  
    """  
  
    fx = f(x) # evaluate function value at original point  
    grad = np.zeros(x.shape)  
    h = 0.00001  
  
    # iterate over all indexes in x  
    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])  
    while not it.finished:  
  
        # evaluate function at x+h  
        ix = it.multi_index  
        old_value = x[ix]  
        x[ix] = old_value + h # increment by h  
        fxh = f(x) # evaluate f(x + h)  
        x[ix] = old_value # restore to previous value (very important!)  
  
        # compute the partial derivative  
        grad[ix] = (fxh - fx) / h # the slope  
        it.iternext() # step to next dimension  
  
    return grad
```

Evaluation the gradient numerically

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

- approximate
- very slow to evaluate

```
def eval_numerical_gradient(f, x):  
    """  
    a naive implementation of numerical gradient of f at x  
    - f should be a function that takes a single argument  
    - x is the point (numpy array) to evaluate the gradient at  
    """  
  
    fx = f(x) # evaluate function value at original point  
    grad = np.zeros(x.shape)  
    h = 0.00001  
  
    # iterate over all indexes in x  
    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])  
    while not it.finished:  
  
        # evaluate function at x+h  
        ix = it.multi_index  
        old_value = x[ix]  
        x[ix] = old_value + h # increment by h  
        fxh = f(x) # evaluate f(x + h)  
        x[ix] = old_value # restore to previous value (very important!)  
  
        # compute the partial derivative  
        grad[ix] = (fxh - fx) / h # the slope  
        it.iternext() # step to next dimension  
  
    return grad
```


This is silly. The loss is just a function of 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 $\nabla_W L$

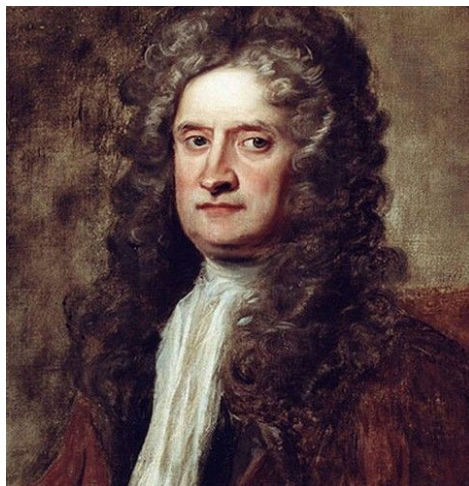
This is silly. The loss is just a function of W :

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

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

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

want $\nabla_W L$



This is silly. The loss is just a function of 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 $\nabla_W L$

Calculus



This is silly. The loss is just a function of W :

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

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

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


$$\nabla_W L = \dots$$

current W:

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

loss 1.25347

$dW = \dots$
(some function
data and W)



gradient dW:

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

In summary:

- Numerical gradient: approximate, slow, easy to write
- Analytic gradient: exact, fast, error-prone

=>

In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check**.

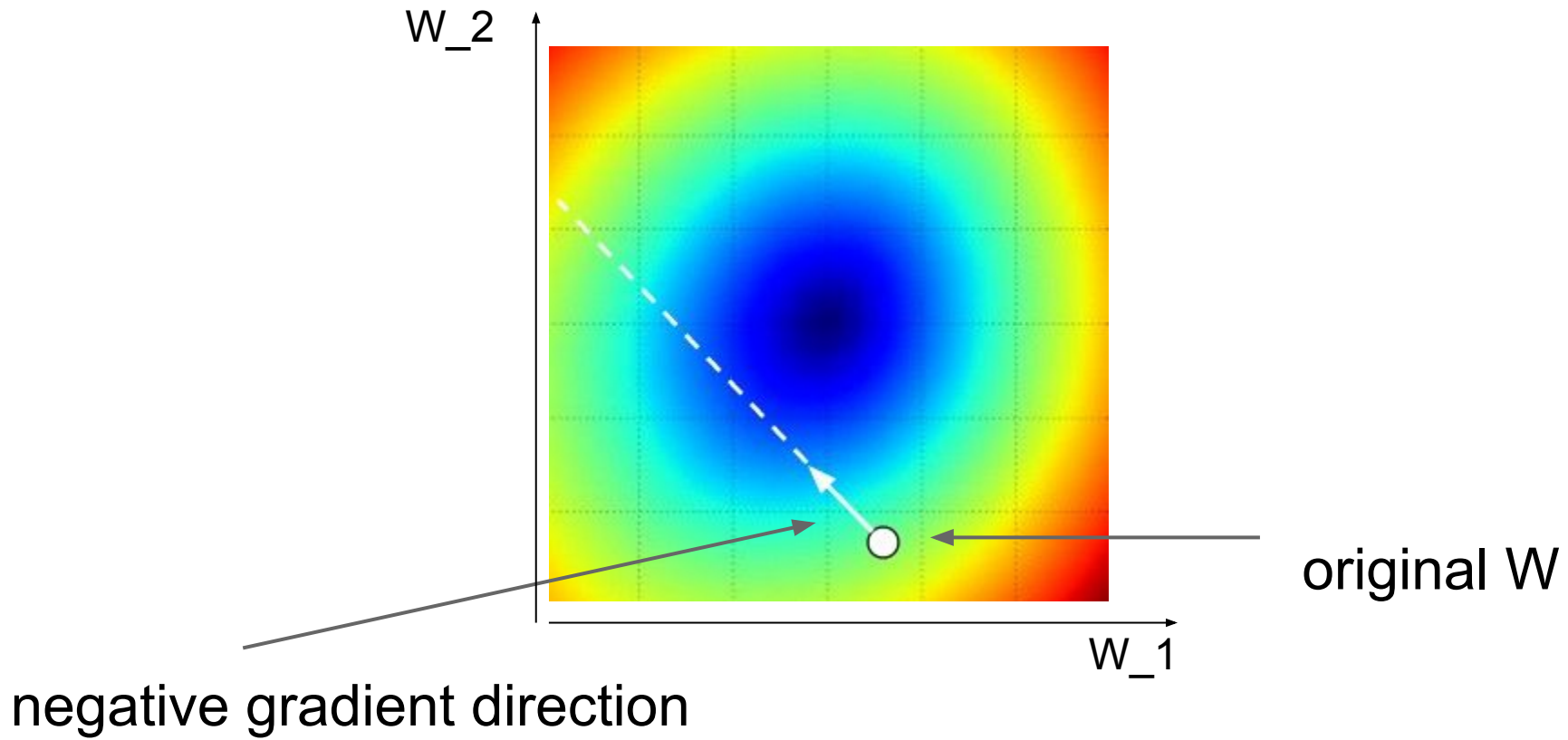
Gradient Descent

```
# Vanilla Gradient Descent
```

```
while True:
```

```
    weights_grad = evaluate_gradient(loss_fun, data, weights)
```

```
    weights += - step_size * weights_grad # perform parameter update
```



Mini-batch Gradient Descent

- only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent
```

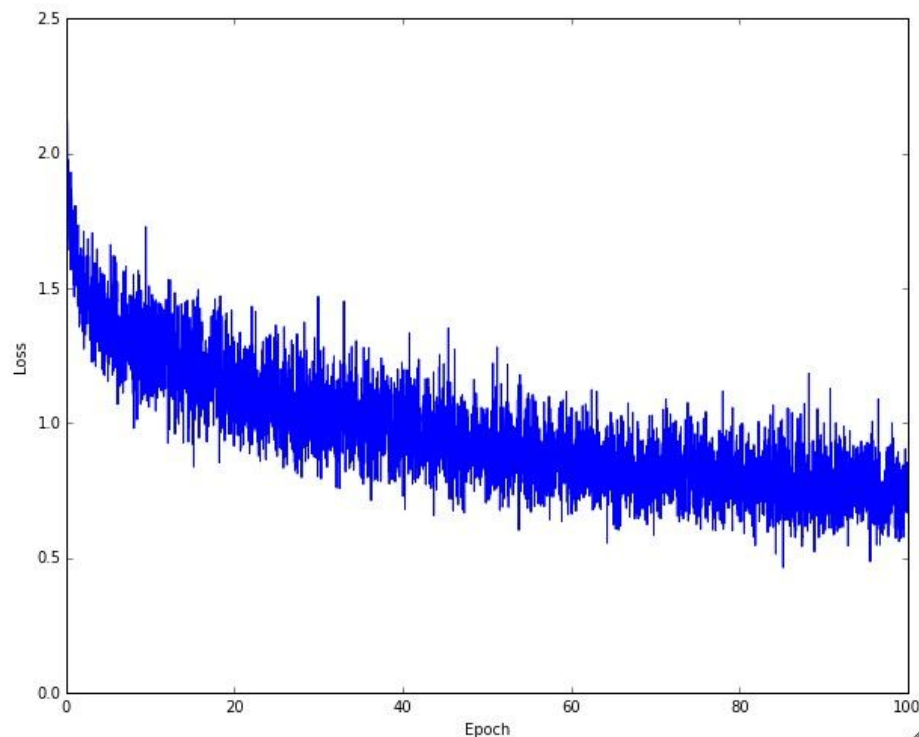
```
while True:
```

```
    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
```

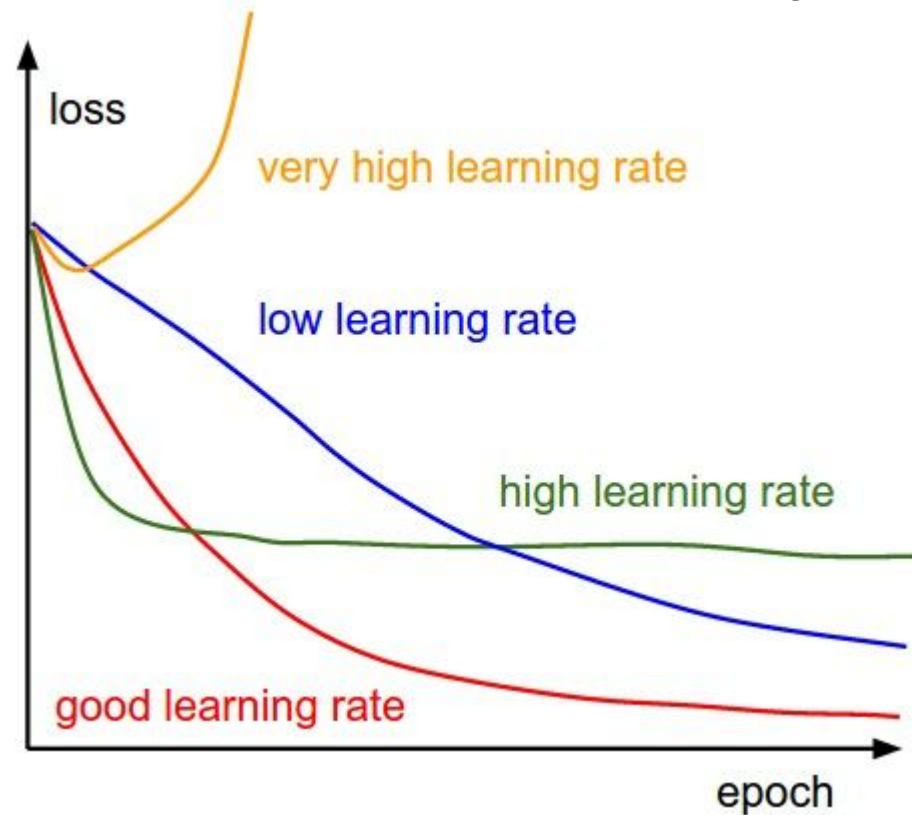
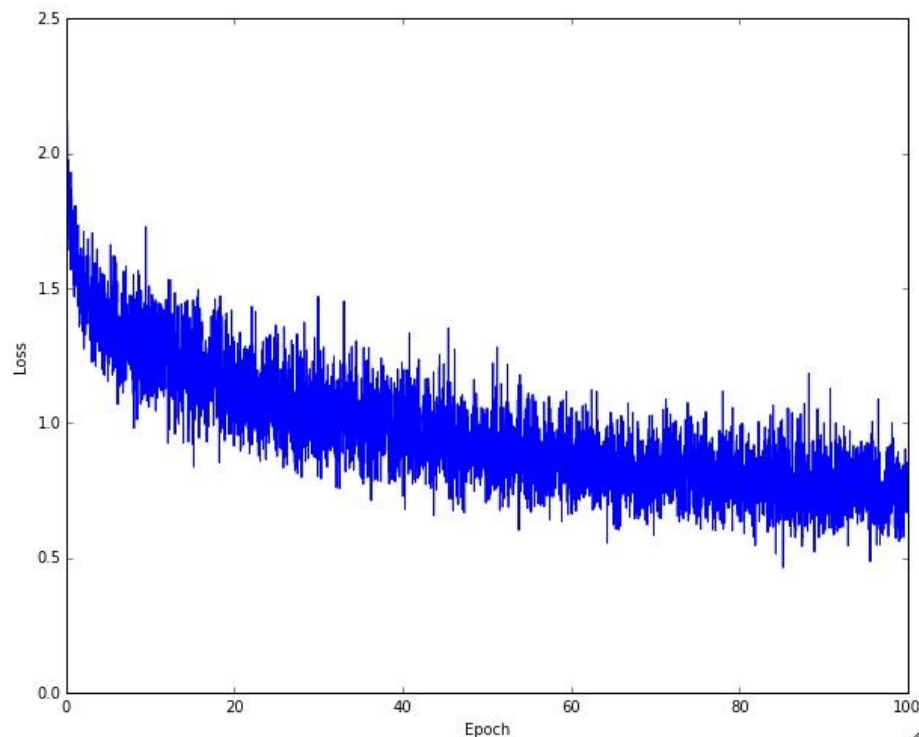
Common mini-batch sizes are 32/64/128 examples
e.g. Krizhevsky ILSVRC ConvNet used 256 examples



Example of optimization progress while training a neural network.

(Loss over mini-batches goes down over time.)

The effects of step size (or “learning rate”)




Mini-batch Gradient Descent

- only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent

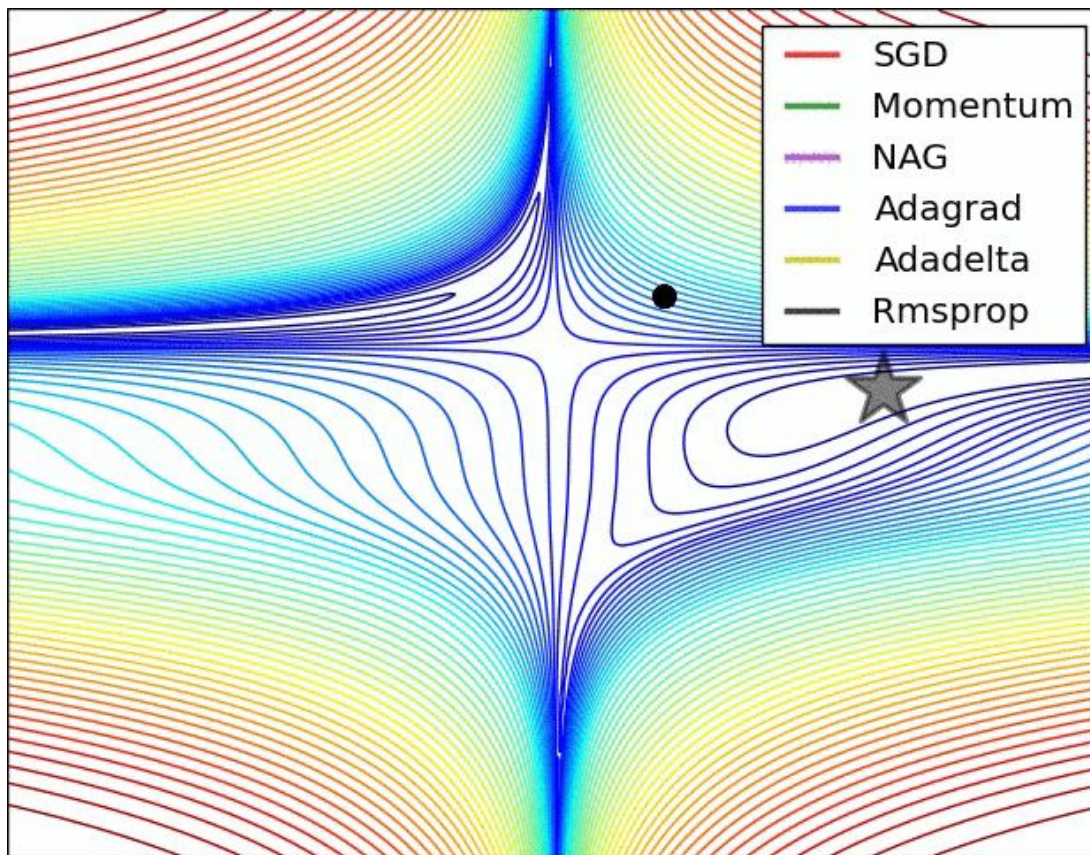
while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

Common mini-batch sizes are 32/64/128 examples
e.g. Krizhevsky ILSVRC ConvNet used 256 examples



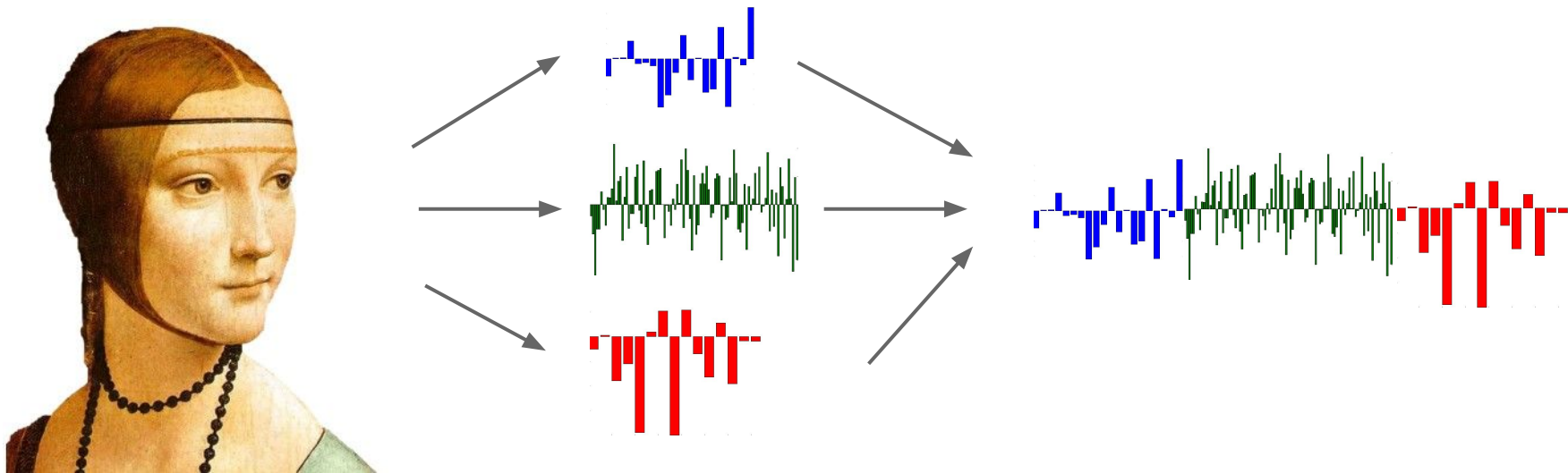
we will look at more
fancy update formulas
(momentum, Adagrad,
RMSProp, Adam, ...)

The effects of different update form formulas

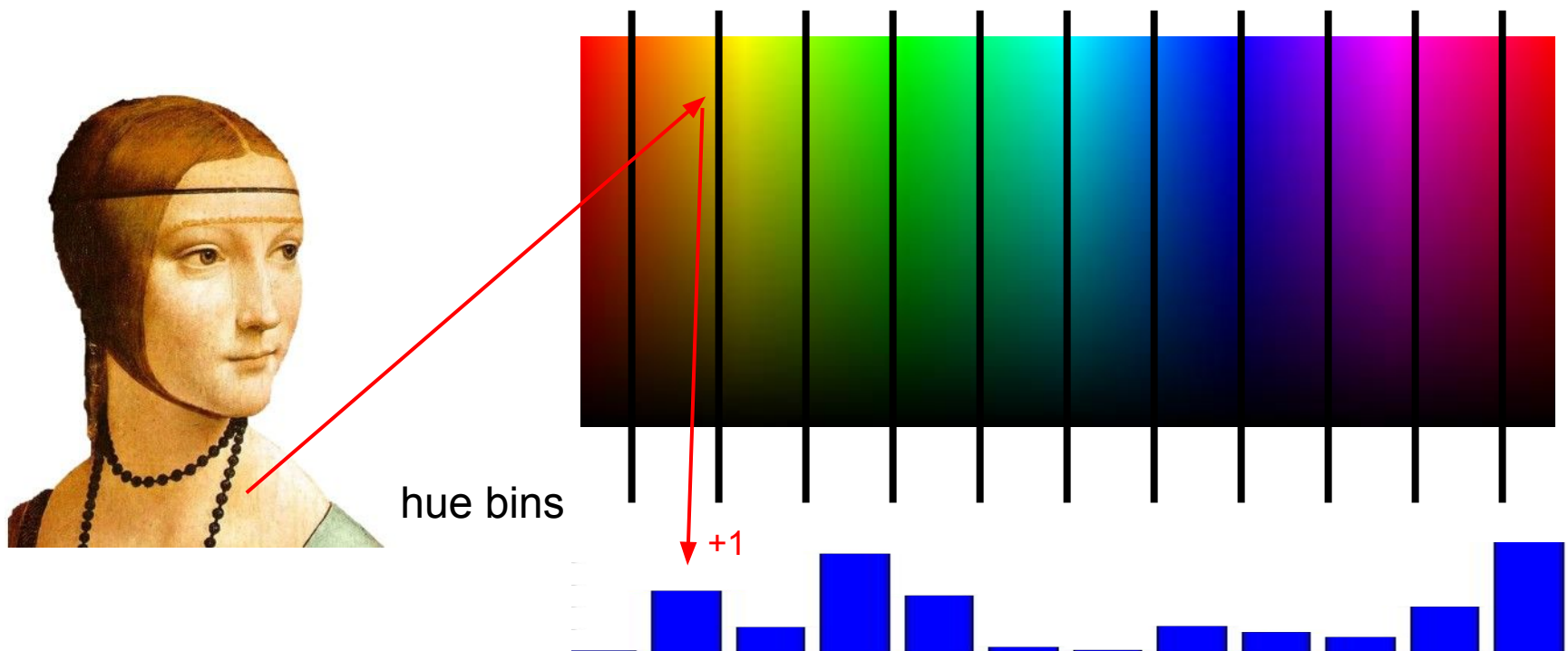


(image credits to Alec Radford)

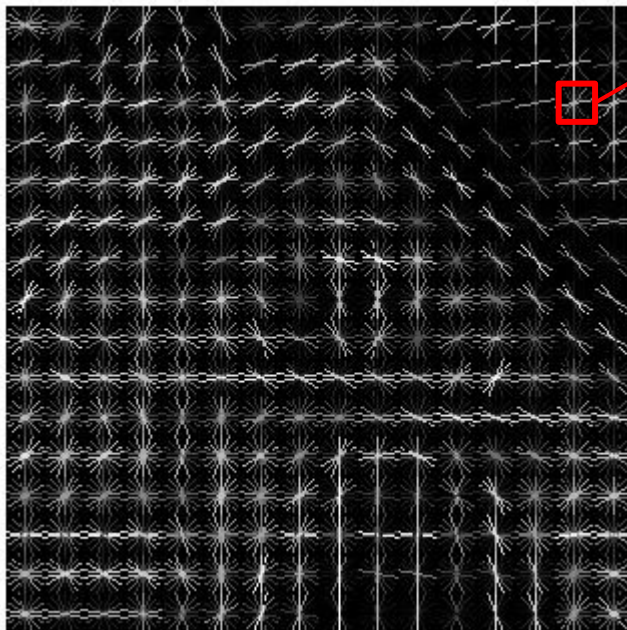
Aside: Image Features



Example: Color (Hue) Histogram



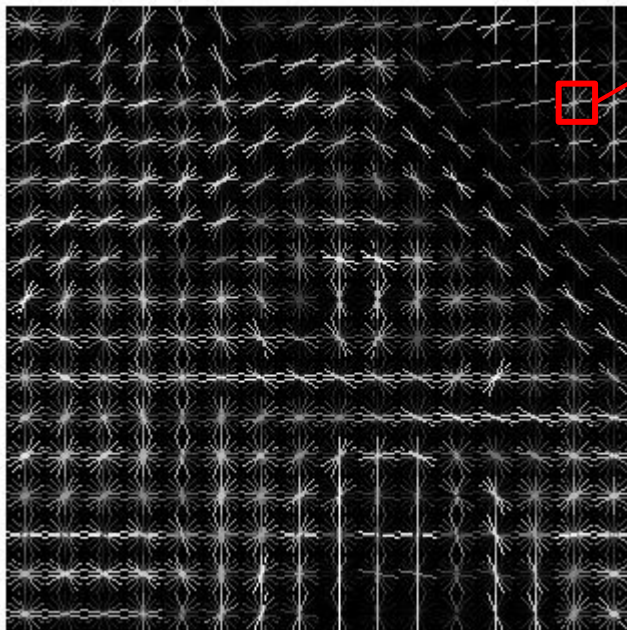
Example: HOG/SIFT features



8x8 pixel region,
quantize the edge
orientation into 9 bins

(image from vlfeat.org)

Example: HOG/SIFT features

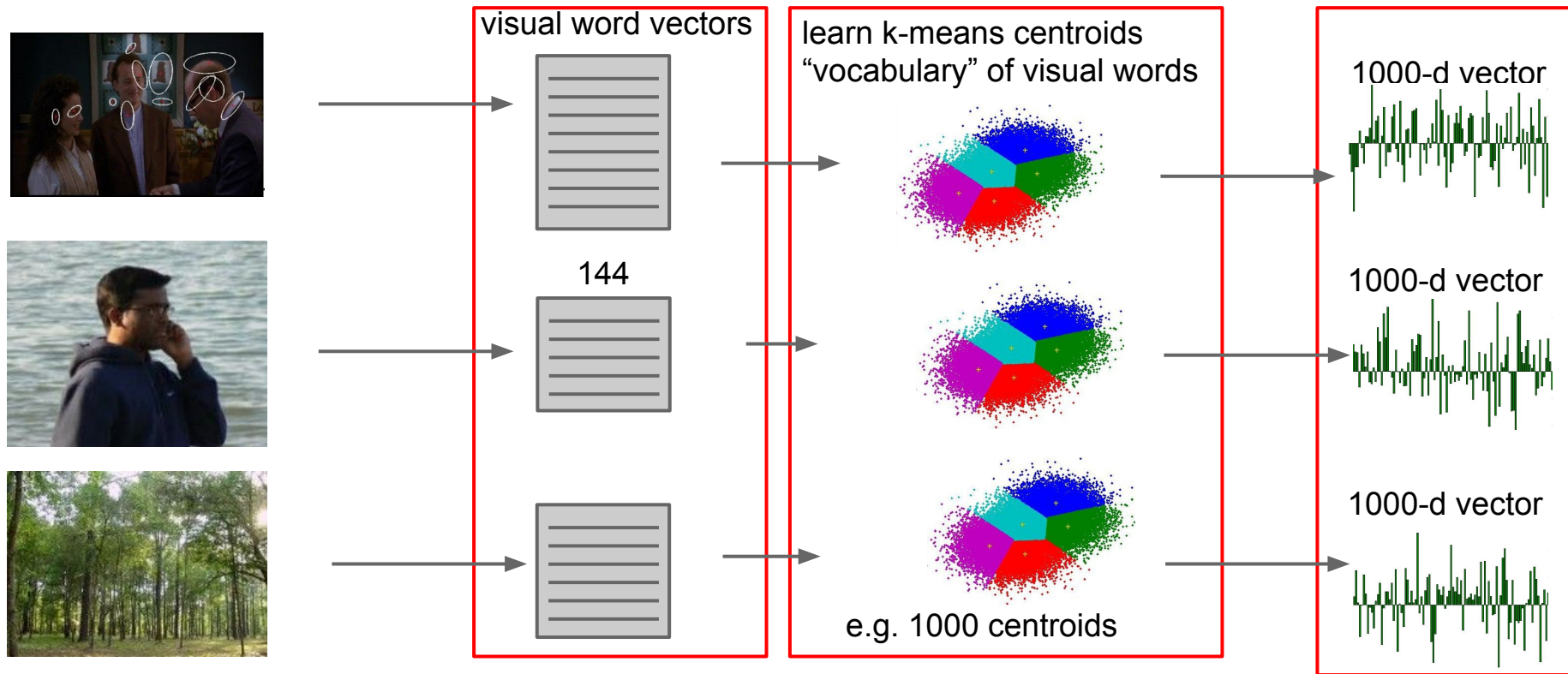


8x8 pixel region,
quantize the edge
orientation into 9 bins

Many more:
GIST, LBP,
Texton,
SSIM, ...

(image from vlfeat.org)

Example: Bag of Words





[32x32x3]



vector describing various
image statistics



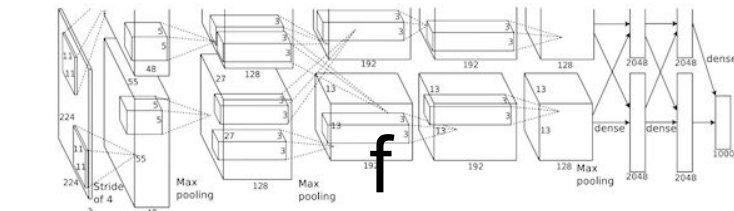
f

training

10 numbers, indicating
class scores



[32x32x3]



training

10 numbers, indicating
class scores

Next class:

Becoming a backprop ninja
and
Neural Networks (part 1)