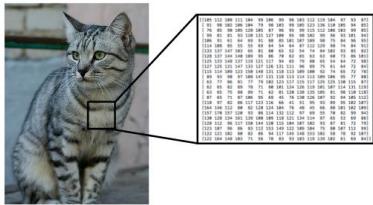


# Lecture 3: Loss Functions and Optimization

# Recall from last time: Challenges of recognition

Viewpoint

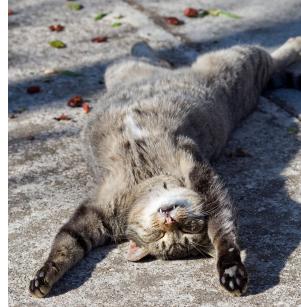


Illumination



[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



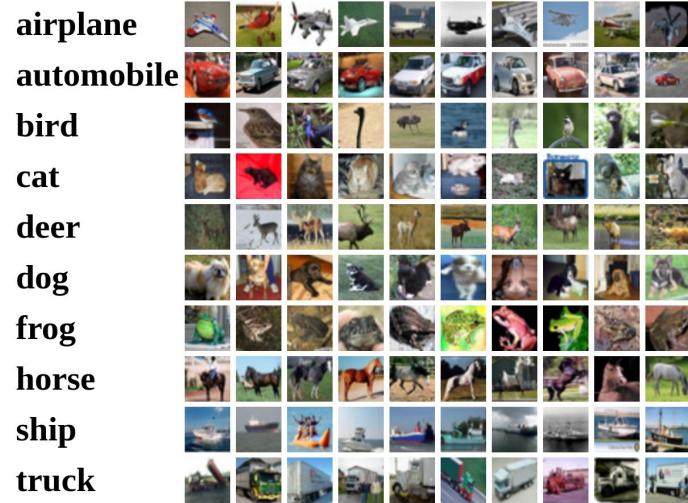
[This image is CC0 1.0 public domain](#)

Intraclass Variation

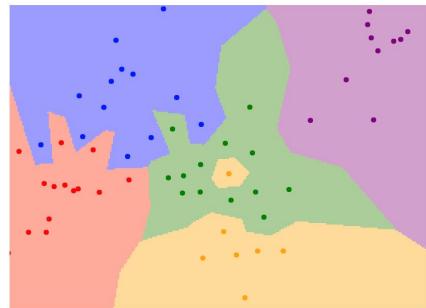


[This image is CC0 1.0 public domain](#)

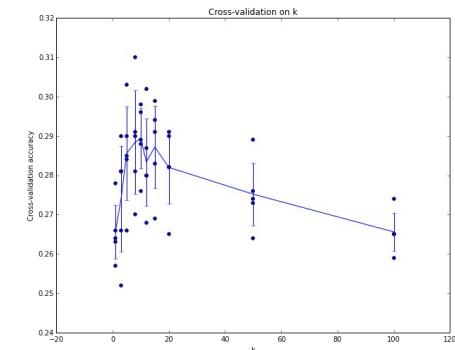
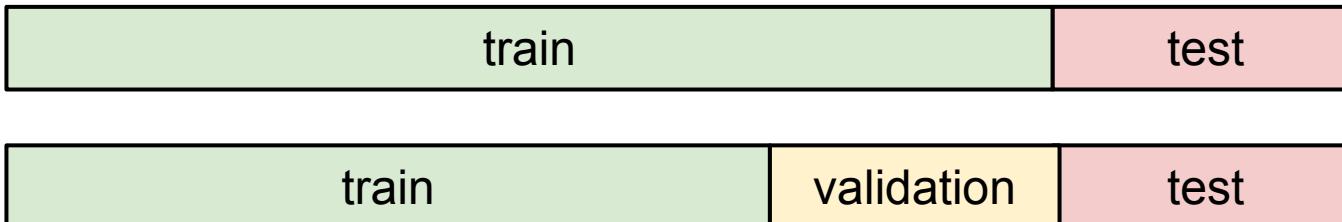
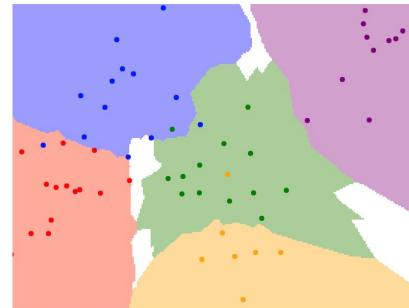
# Recall from last time: data-driven approach, kNN



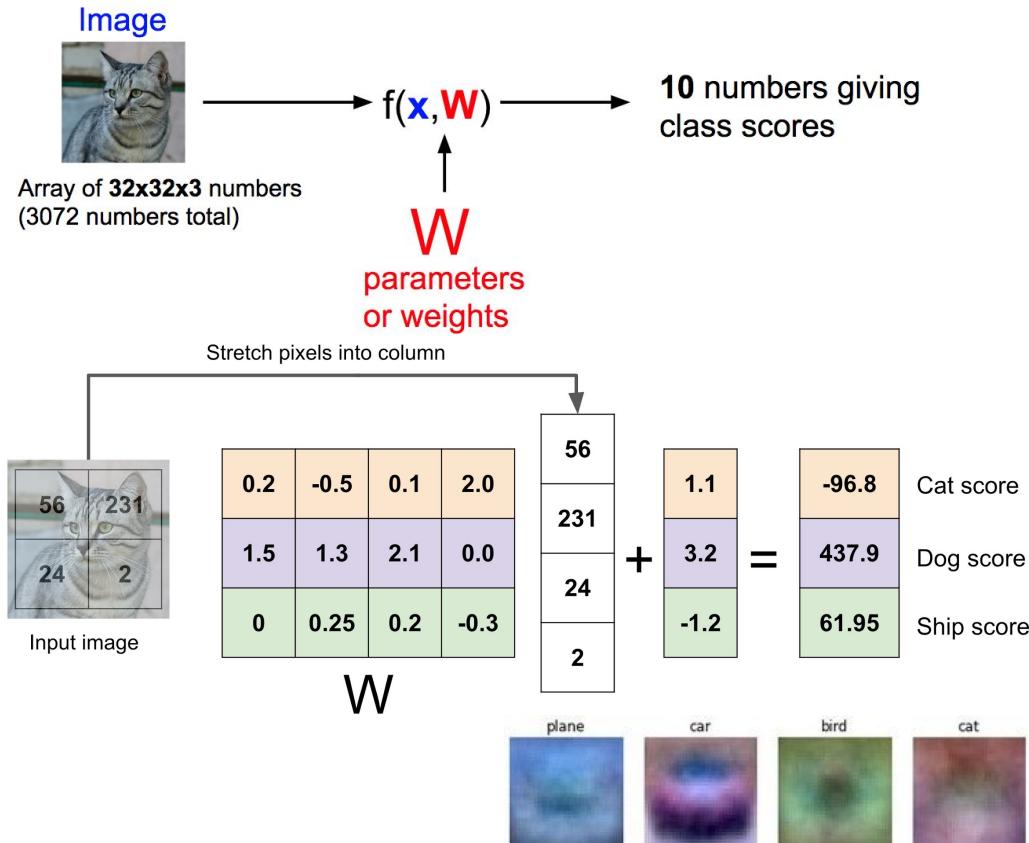
1-NN classifier



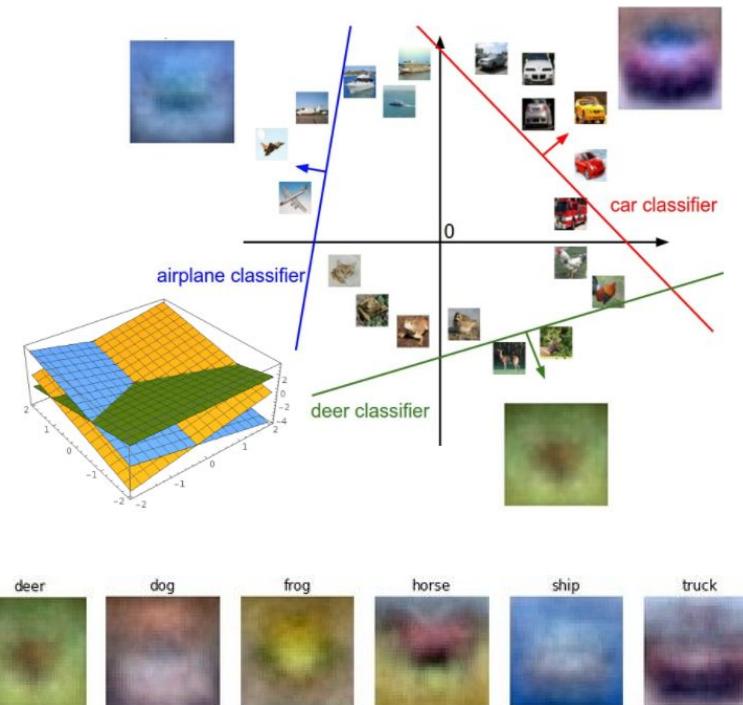
5-NN classifier



# Recall from last time: Linear Classifier



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



# Recall from last time: Linear Classifier



airplane	-3.45	-0.51	3.42
automobile	-8.87	<b>6.04</b>	4.64
bird	0.09	5.31	2.65
cat	<b>2.9</b>	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	<b>-4.34</b>
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)**

Cat image by Nikita is licensed under CC-BY 2.0; Car image is CC0 1.0 public domain; Frog image is in the public domain

Suppose: 3 training examples, 3 classes.

With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	<b>3.2</b>	1.3	2.2
car	<b>5.1</b>	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

Suppose: 3 training examples, 3 classes.  
With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

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:

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

Suppose: 3 training examples, 3 classes.  
With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

## 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:

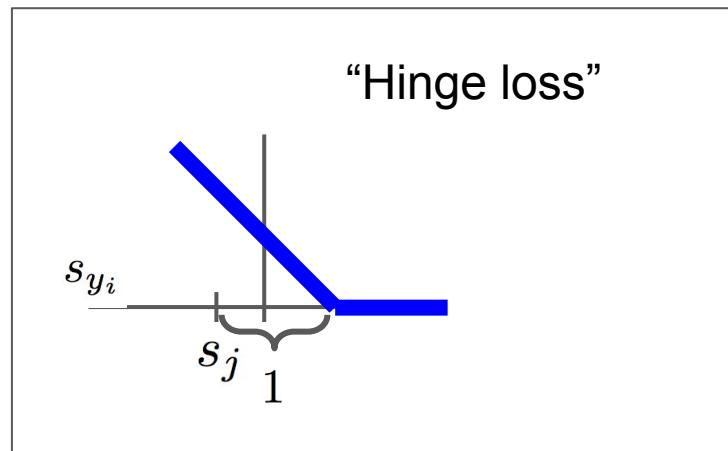
$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Suppose: 3 training examples, 3 classes.  
With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

## Multiclass SVM loss:



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

Suppose: 3 training examples, 3 classes.  
With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

## 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	<b>3.2</b>	1.3	2.2
car	<b>5.1</b>	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	<b>2.9</b>		

## 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	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
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	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	2.9	0	<b>12.9</b>

## 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:

$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\
 &= \max(0, 2.2 - (-3.1) + 1) \\
 &\quad + \max(0, 2.5 - (-3.1) + 1) \\
 &= \max(0, 6.3) + \max(0, 6.6) \\
 &= 6.3 + 6.6 \\
 &= 12.9
 \end{aligned}$$

Suppose: 3 training examples, 3 classes.  
With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	<b>2.9</b>	0	<b>12.9</b>

## 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:

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

Loss over full dataset is average:

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

$$\begin{aligned} L &= (2.9 + 0 + 12.9)/3 \\ &= 5.27 \end{aligned}$$

# 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
    loss_i = np.sum(margins)
    return loss_i
```

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

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

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

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

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

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

**No!  $2W$  is also has  $L = 0!$**

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:**

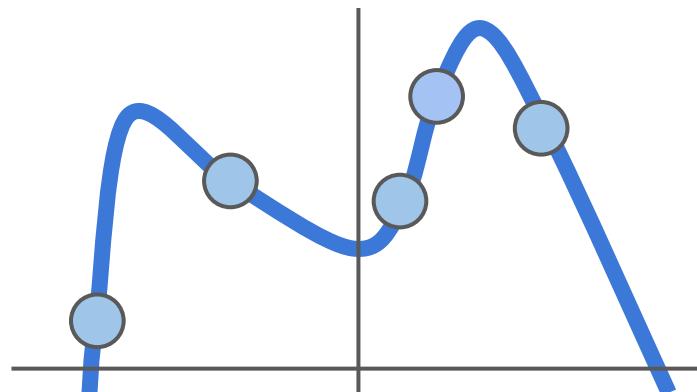
$$\begin{aligned} &= \max(0, 2.6 - 9.8 + 1) \\ &\quad + \max(0, 4.0 - 9.8 + 1) \\ &= \max(0, -6.2) + \max(0, -4.8) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

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


**Data loss:** Model predictions  
should match training data

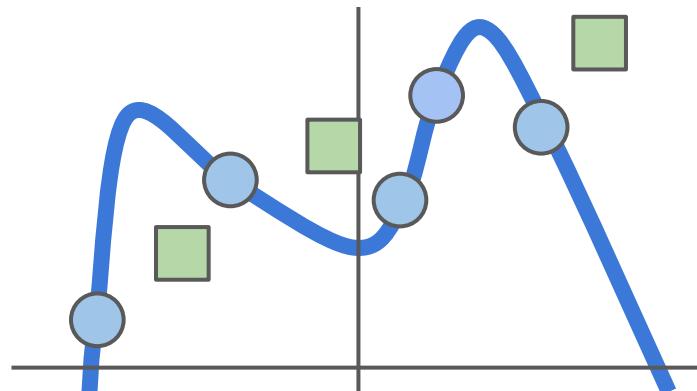
$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)$$

**Data loss:** Model predictions  
should match training data



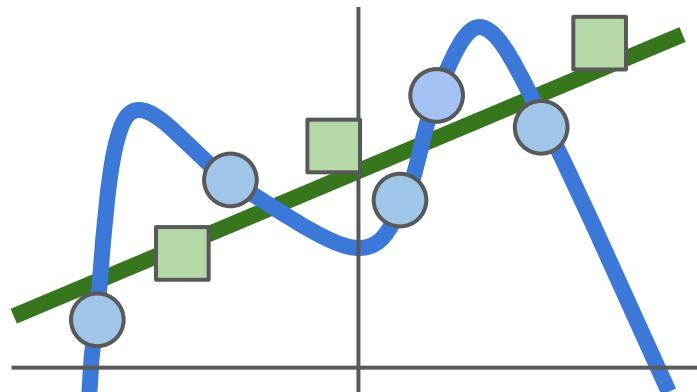
$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)$$

**Data loss:** Model predictions  
should match training data



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

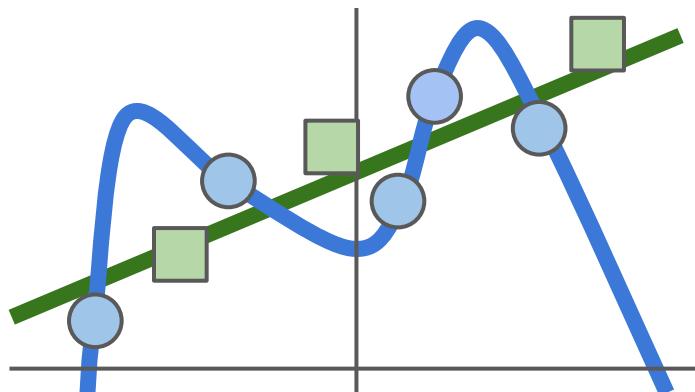
**Data loss:** Model predictions  
should match training data



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

**Data loss:** Model predictions should match training data

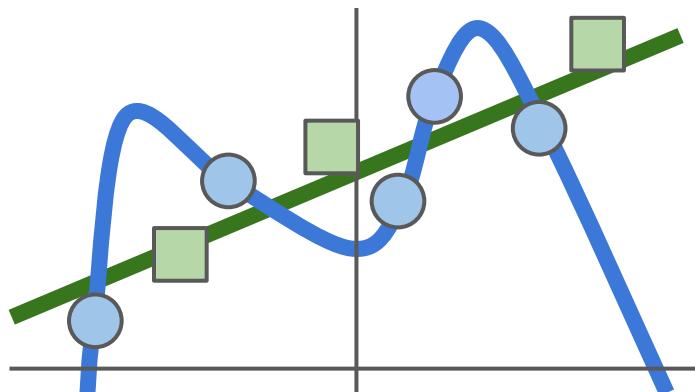
**Regularization:** Model should be “simple”, so it works on test data



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

**Data loss:** Model predictions should match training data

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



**Occam's Razor:**  
*“Among competing hypotheses,  
the simplest is the best”*  
William of Ockham, 1285 - 1347

# Regularization

$\lambda$  = regularization strength  
(hyperparameter)

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

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 Regularization (Weight Decay)

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

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

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

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

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

# L2 Regularization (Weight Decay)

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

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

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

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

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

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

# Softmax Classifier (Multinomial Logistic Regression)



cat	<b>3.2</b>
car	5.1
frog	-1.7

# Softmax Classifier (Multinomial Logistic Regression)



**scores = unnormalized log probabilities of the classes.**

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

cat	<b>3.2</b>
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	<b>3.2</b>
car	<b>5.1</b>
frog	<b>-1.7</b>

# Softmax Classifier (Multinomial Logistic Regression)



**scores = unnormalized log probabilities of the classes.**

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

cat	3.2	Softmax function
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	<b>3.2</b>
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:

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

# 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	<b>3.2</b>
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:  $L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$

# Softmax Classifier (Multinomial Logistic Regression)



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

cat  
car  
frog

3.2
5.1
-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

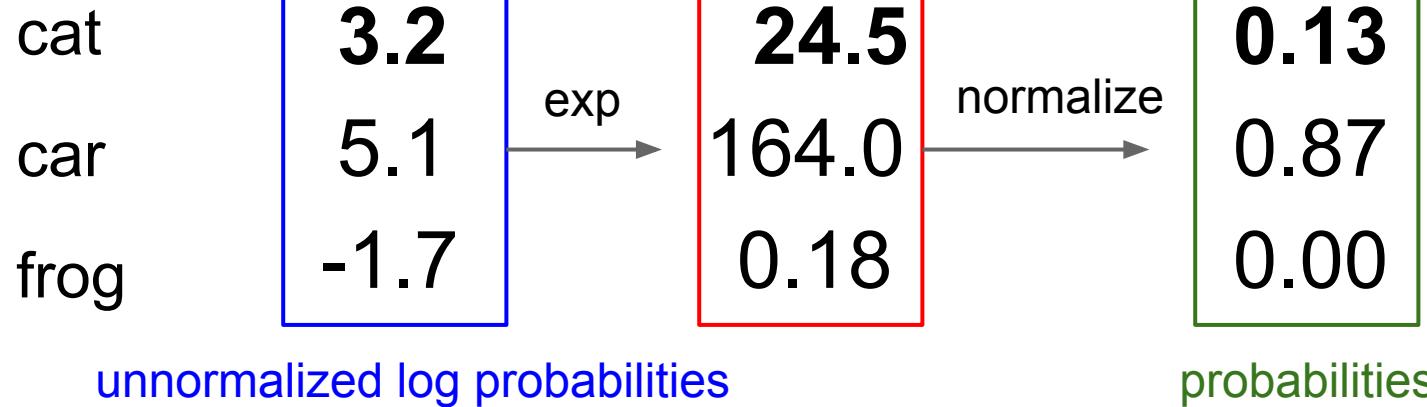
unnormalized log probabilities

# Softmax Classifier (Multinomial Logistic Regression)



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

unnormalized 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

$$\begin{aligned} L_i &= -\log(0.13) \\ &= 0.89 \end{aligned}$$

unnormalized log probabilities

probabilities

hinge loss (SVM)

matrix multiply + bias offset

0.01	-0.05	0.1	0.05
0.7	0.2	0.05	0.16
0.0	-0.45	-0.2	0.03

$W$

-15	0.0
22	0.2
-44	-0.3
56	

$x_i$

$y_i$  2

+

$b$

-2.85
0.86
0.28

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

cross-entropy loss (Softmax)

-2.85	0.058	0.016
0.86	2.36	0.631
0.28	1.32	0.353

$\exp$

normalize  
(to sum to one)

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

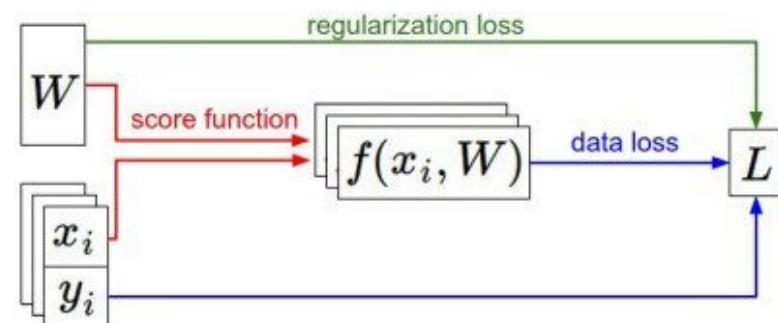
# Recap

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

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

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

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

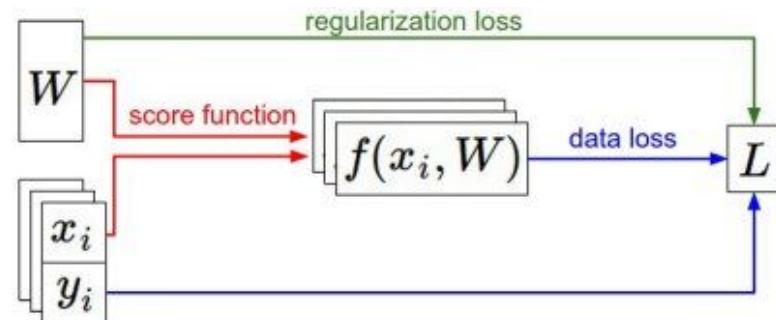


# Recap

## How do we find the best $W$ ?

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

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right) \quad \text{Softmax}$$
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad \text{SVM}$$
$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W) \quad \text{Full loss}$$



# Optimization



This image is CC0 1.0 public domain



[Walking man image is CC0 1.0 public domain](#)

# 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



## Strategy #2: Follow the slope

In 1-dimension, the derivative of a function:

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

Text

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

The direction of steepest descent is the **negative gradient**

**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,**

? ,

? ,

$$\frac{(1.25322 - 1.25347)}{0.0001} = -2.5$$

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

? ,

? , ... ]

current W:	W + h (second dim):	gradient dW:
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]	[0.34, -1.11 + <b>0.0001</b> , 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]	[-2.5, ?, ?, ?, ?, ?, ?, ?, ?, ?,...]

**loss 1.25347**      **loss 1.25353**

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,**  
?,  
?,

$$\frac{(1.25353 - 1.25347)}{0.0001} = 0.6$$

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

?,...]

current W:	W + h (third dim):	gradient dW:
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]	[0.34, -1.11, 0.78 + <b>0.0001</b> , 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]	[-2.5, 0.6, ?, ?, ?, ?, ?, ?, ?, ?,...]

**loss 1.25347**      **loss 1.25347**

**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**,  
?,  
?]

$$\frac{(1.25347 - 1.25347)}{0.0001} = 0$$

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

?, ...]

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  $\nabla_W L$

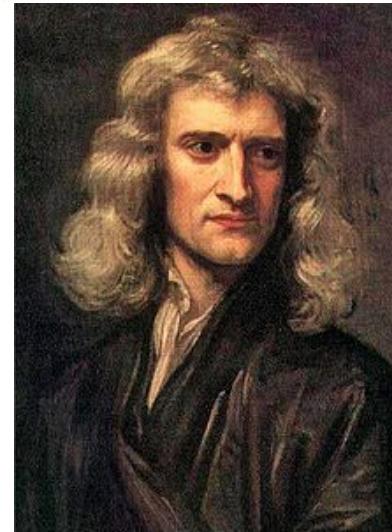
# This is silly. The loss is just a function of W:

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

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

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

want  $\nabla_W L$



[This image is in the public domain](#)



[This image is in the public domain](#)

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

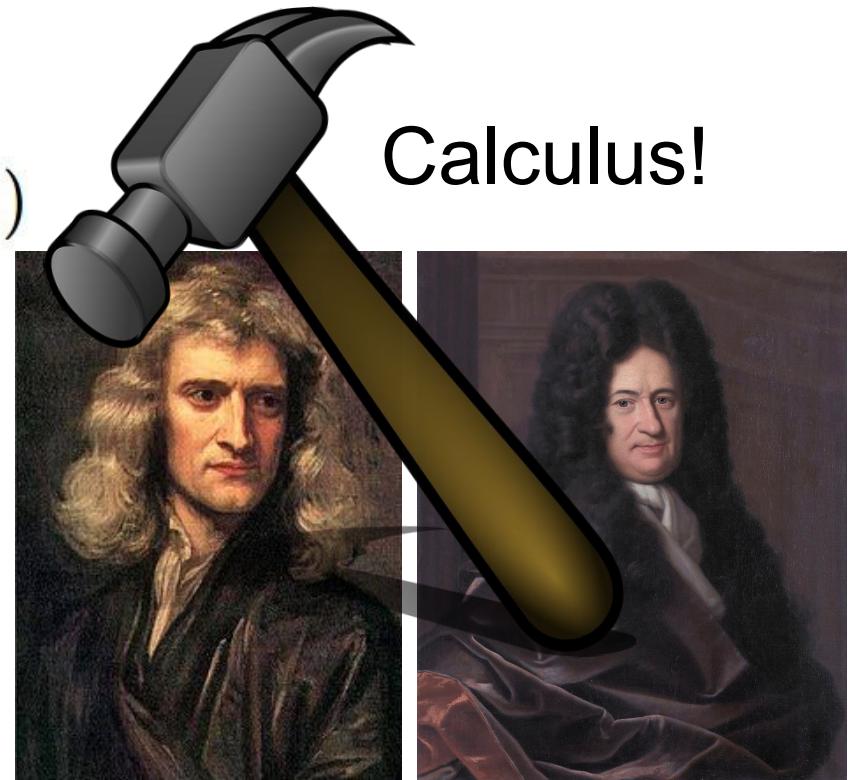
$$L = \frac{1}{N} \sum_{i=1}^N L_i + \sum_k W_k^2$$

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

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

want  $\nabla_W L$

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**

**gradient dW:**

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

dW = ...  
(some function  
data and W)



# In summary:

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

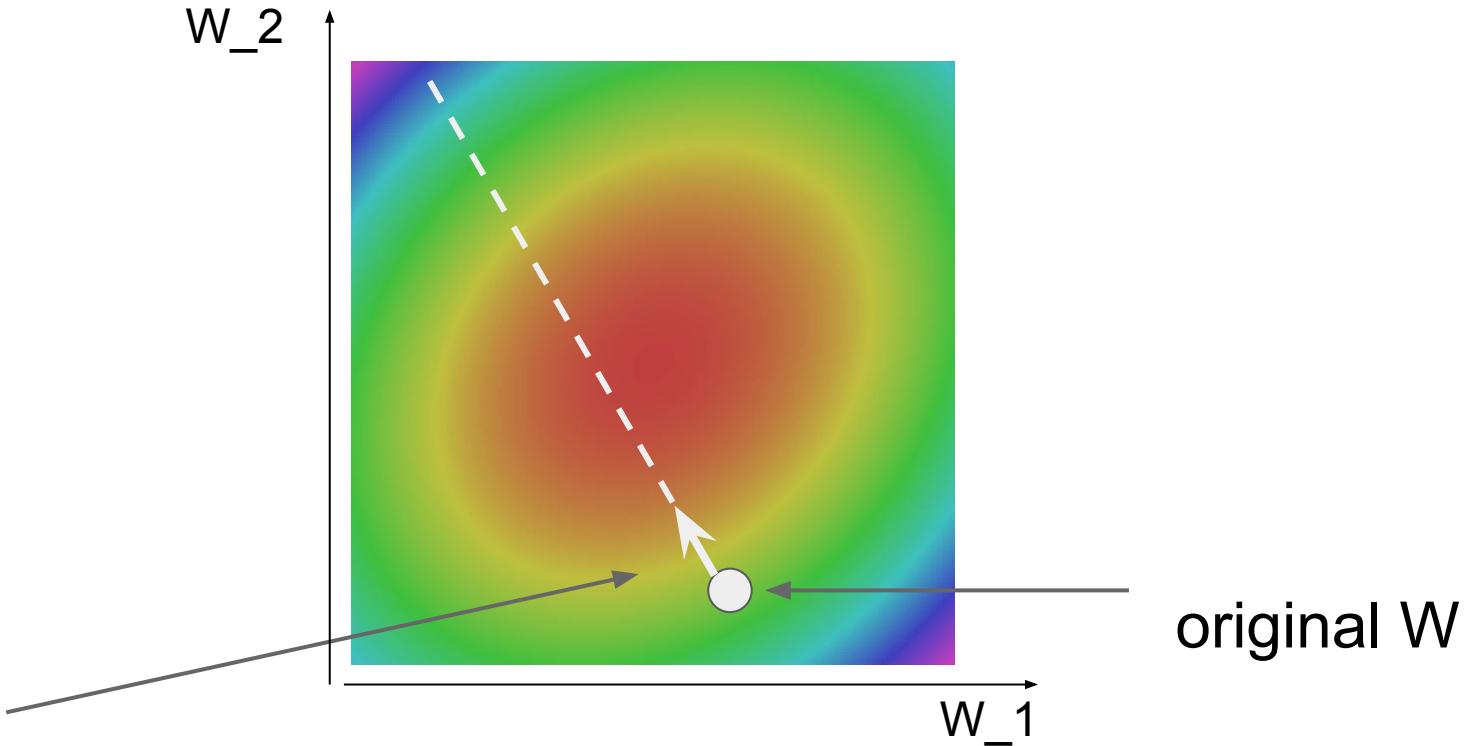
=>

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

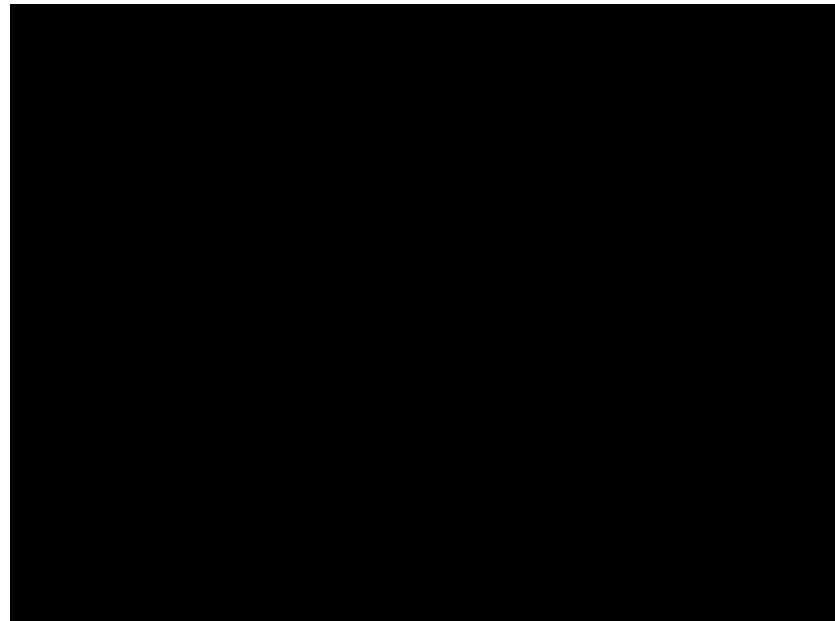
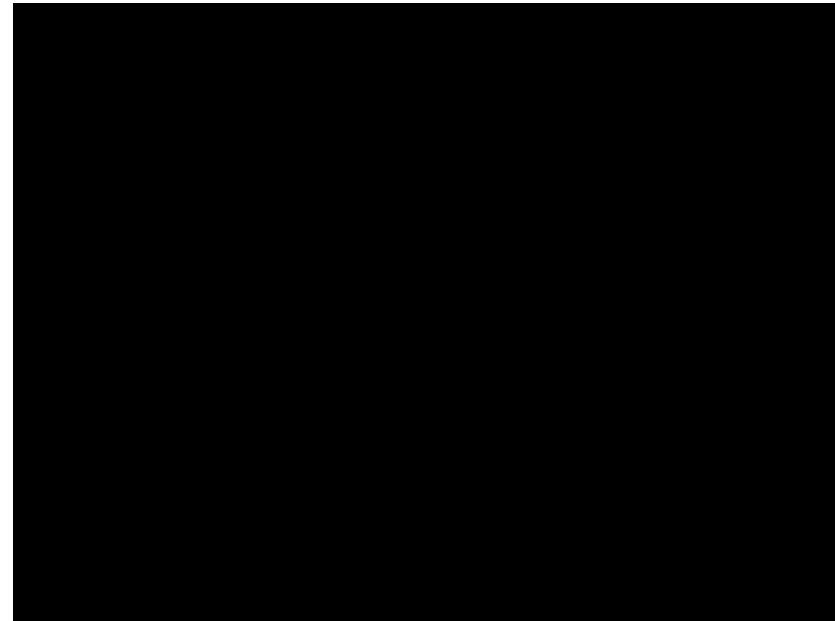
# Gradient Descent

```
# Vanilla Gradient Descent

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



negative gradient direction



# Stochastic Gradient Descent (SGD)

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

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

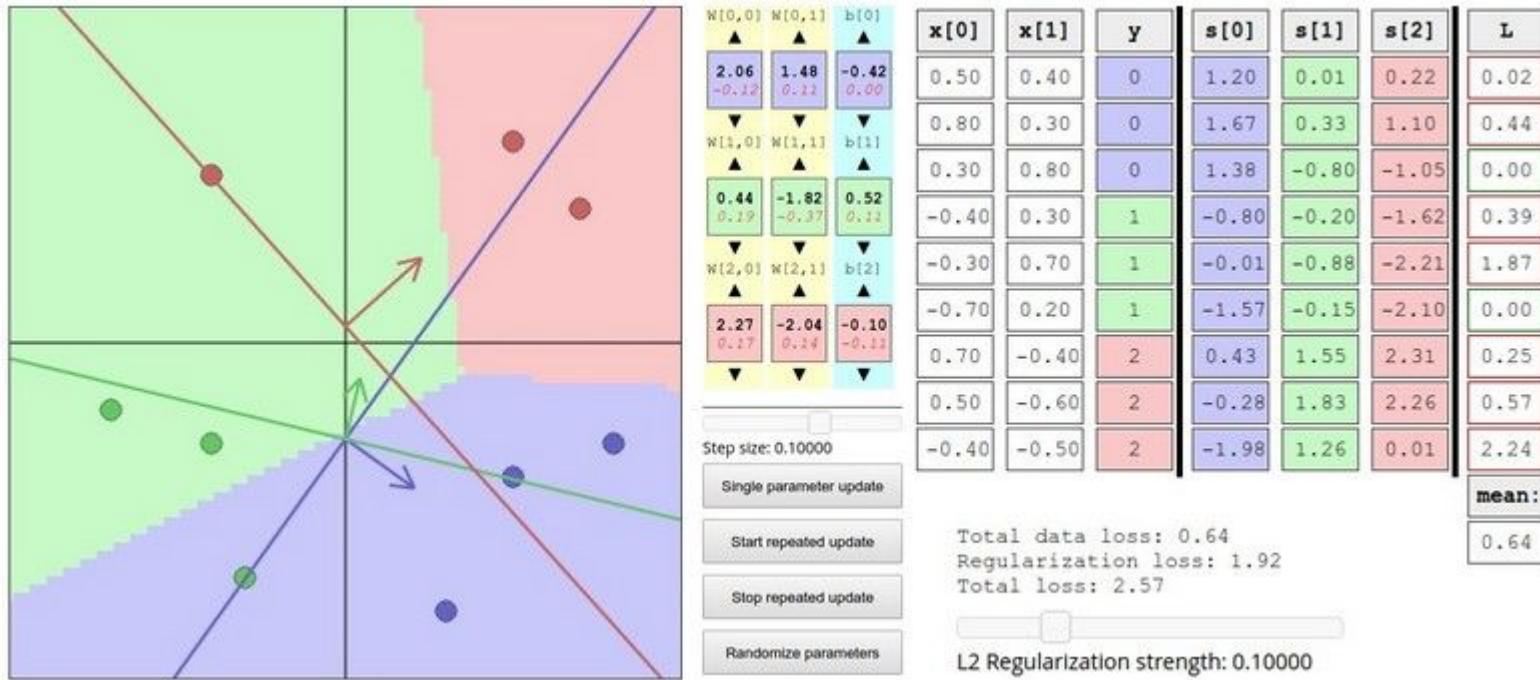
Full sum expensive  
when N is large!

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

```
# 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
```

# Interactive Web Demo time....



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

# Interactive Web Demo time....

