

Lecture 3: Loss Functions and Optimization

Announcements: Assignment 1

Released last week, due **Fri 4/16 at 11:59pm**

Announcements: Midterm details

- Tues, **May 4** and is worth **15%** of your grade.
- available for **24 hours** on Gradescope from May 4, **12PM PDT** to May 5, 11:59 AM PDT.
- **3-hour** consecutive timeframe
- Exam will be designed for **1.5 hours**.
- Open book and open internet but no collaboration
- Only make private posts during those 24 hours

Announcements: Project proposal

Due **Mon 4/19**

TA expertise are posted on the webpage.

(http://cs231n.stanford.edu/office_hours.html)

Administrative: Piazza

Please make sure to check and read all pinned piazza posts.

Image Classification: A core task in Computer Vision



This image by [Nikita](#) is
licensed under [CC-BY 2.0](#)

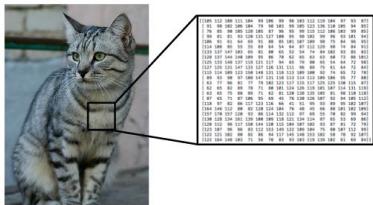
(assume given a set of labels)
{dog, cat, truck, plane, ...}



cat
dog
bird
deer
truck

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 ionsson is licensed
under CC-BY 2.0

Clutter



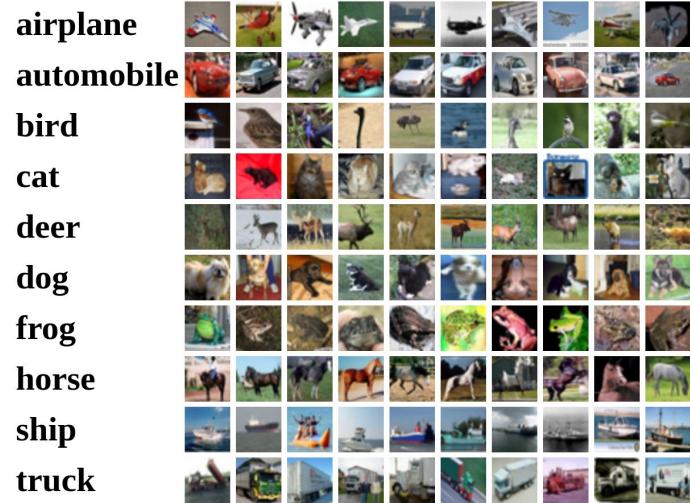
[This image](#) is CC0 1.0 public domain

Intraclass Variation

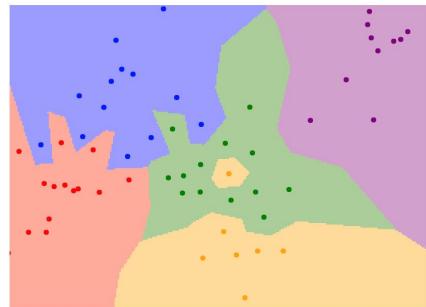


[This image](#) is CC0 1.0 public domain

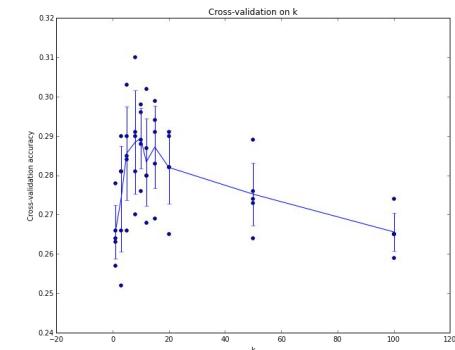
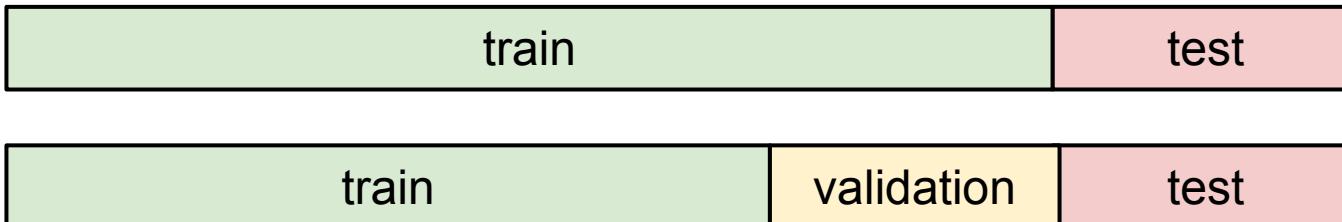
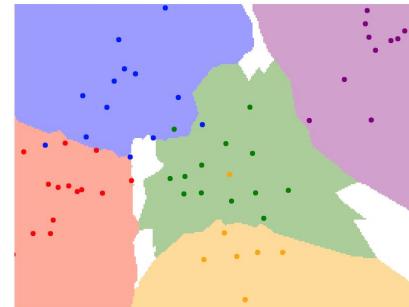
Recall from last time: data-driven approach, kNN



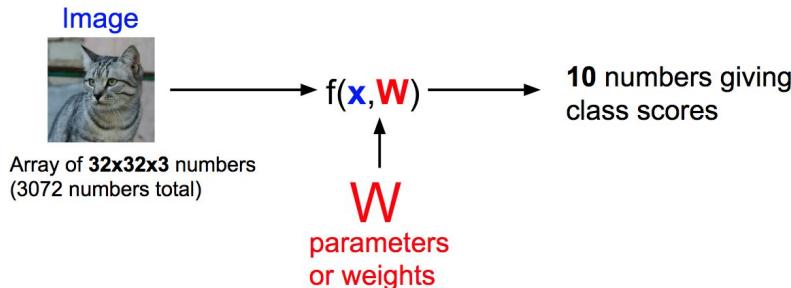
1-NN classifier



5-NN classifier



Recall from last time: Linear Classifier



$$f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Algebraic Viewpoint

$$f(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x}$$

Stretch pixels into column:

Input image

W

b

56	231	2				
24	1	2				
0.2	-0.5	0.1	2.0	56	1.1	-96.8
1.5	1.3	2.1	0.0	231	3.2	437.9
0	0.25	0.2	-0.3	24	-1.2	61.95

Cat score
Dog score
Ship score

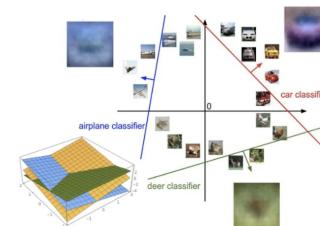
Visual Viewpoint

One template per class



Geometric Viewpoint

Hyperplanes cutting up space

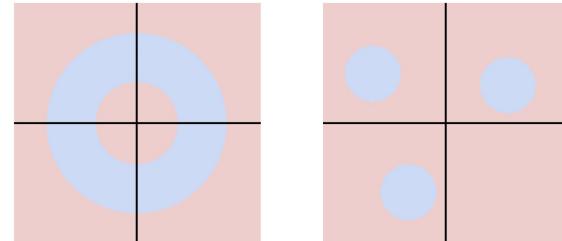


Class 1:
 $1 \leq L_2 \text{ norm} \leq 2$

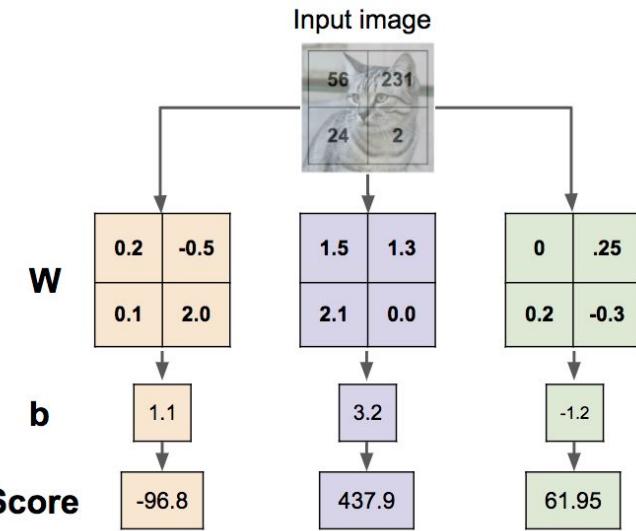
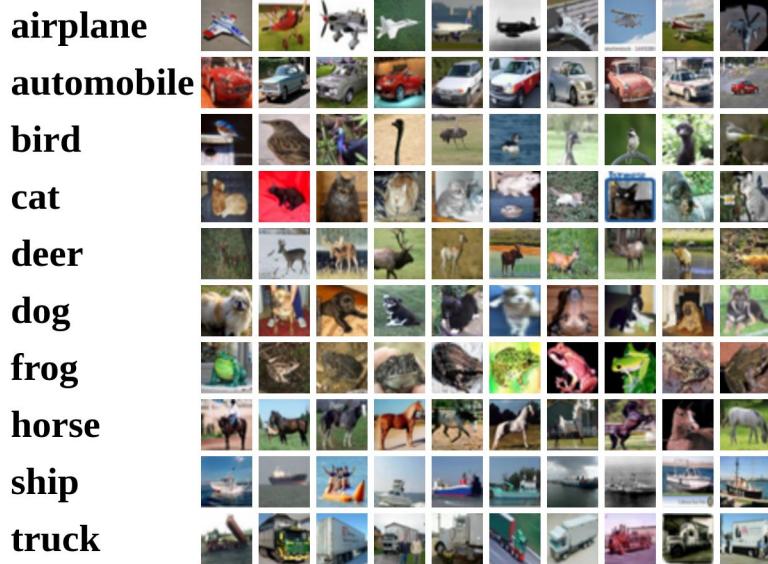
Class 2:
Everything else

Class 1:
Three modes

Class 2:
Everything else



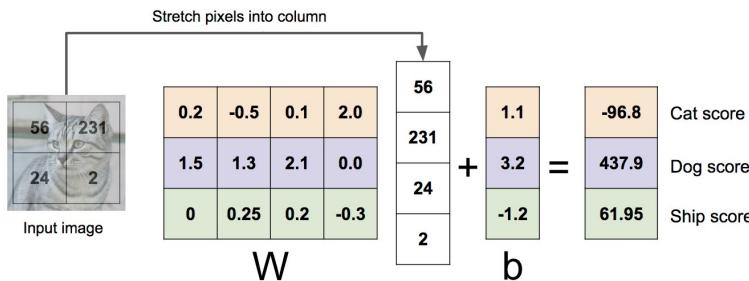
Interpreting a Linear Classifier: Visual Viewpoint



Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

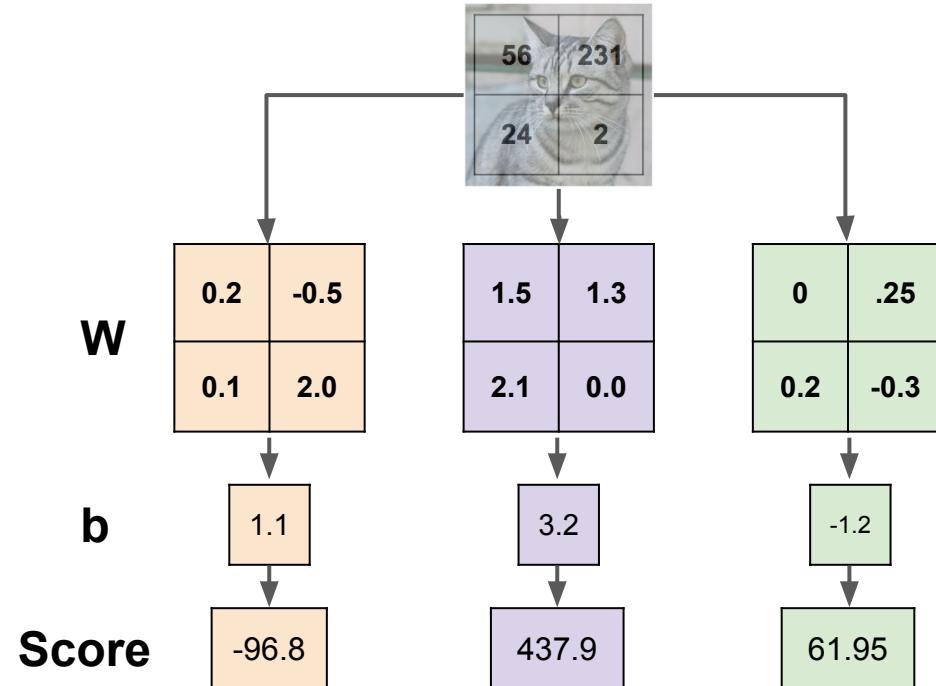
Algebraic Viewpoint

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

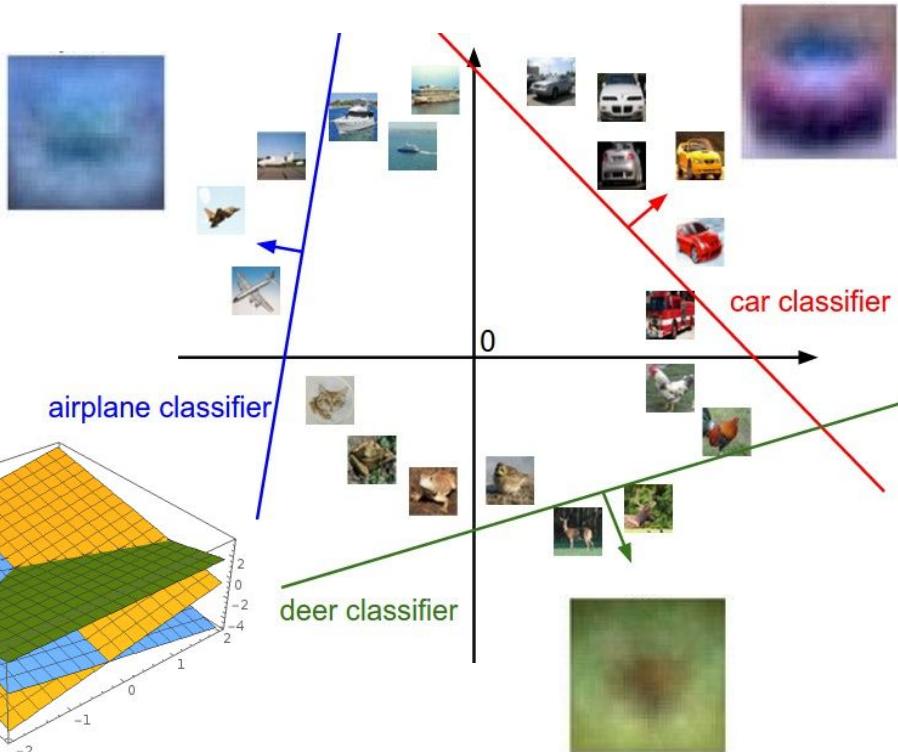


Visual Viewpoint

Input image



Interpreting a Linear Classifier: Geometric Viewpoint



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



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

Plot created using [Wolfram Cloud](#)

[Cat image](#) by [Nikita](#) is licensed under [CC-BY 2.0](#)

Recall from last time: Linear Classifier



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:

定义损失函数训练线性网络
完成多分类任务，最后用
softmax表示概率建立线性分
类器

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.
1. 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	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:

A loss function tells how good our current classifier is



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

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

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 average 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	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:

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

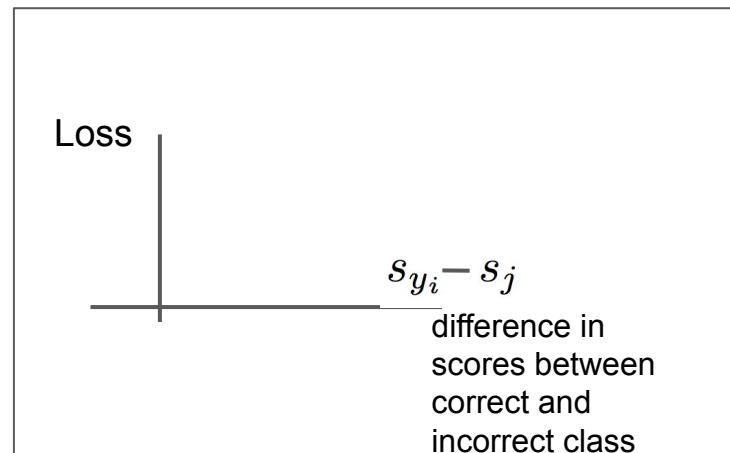
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

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



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

Interpreting Multiclass SVM loss:



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

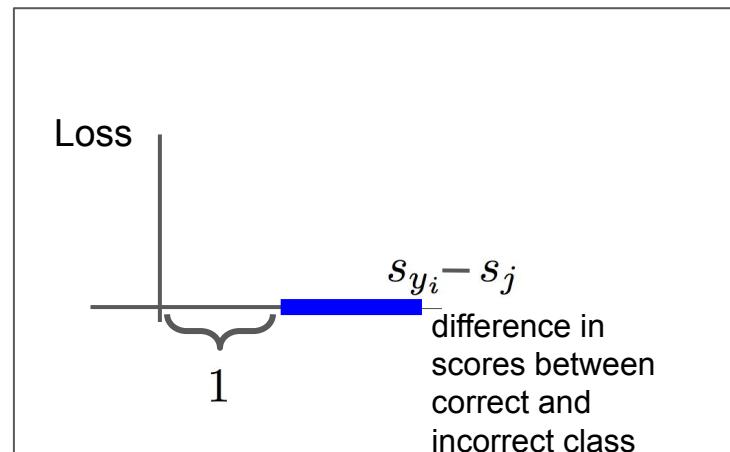
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

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



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

Interpreting Multiclass SVM loss:



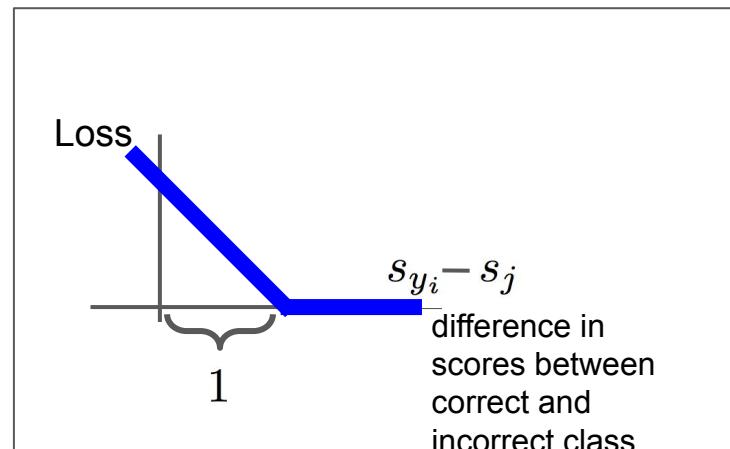
$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases} \\
 &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
 \end{aligned}$$

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



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

Interpreting 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	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: $s = f(x_i, W)$

线性变换得到的矩阵向量相乘结果作为评价指标

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	12.9

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:

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

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



cat	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

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:

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

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

Suppose: 3 training examples, 3 classes.

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

Multiclass SVM loss:

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



cat	1.3
car	4.9
frog	2.0
Losses:	0

Q1: What happens to loss if car scores decrease by 0.5 for this training example?

Q2: what is the min/max possible SVM loss L_i ?

Q3: At initialization W is small so all $s \approx 0$. What is the loss L_i , assuming N examples and C classes?

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	12.9

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 if the sum was 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	12.9

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:

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

Q5: What if we used mean instead of sum?

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	12.9

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:

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

Q6: What if we used

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

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)                                     # First calculate scores
    margins = np.maximum(0, scores - scores[y] + 1)      # Then calculate the margins  $s_j - s_{y_i} + 1$ 
    margins[y] = 0                                         # only sum j is not  $y_i$ , so when  $j = y_i$ , set to zero.
    loss_i = np.sum(margins)                               # sum across all j
    return loss_i
```

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

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

Q7. 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:

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

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

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

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

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

How do we choose between W and $2W$?

Regularization

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

Data loss: Model predictions
should match training data

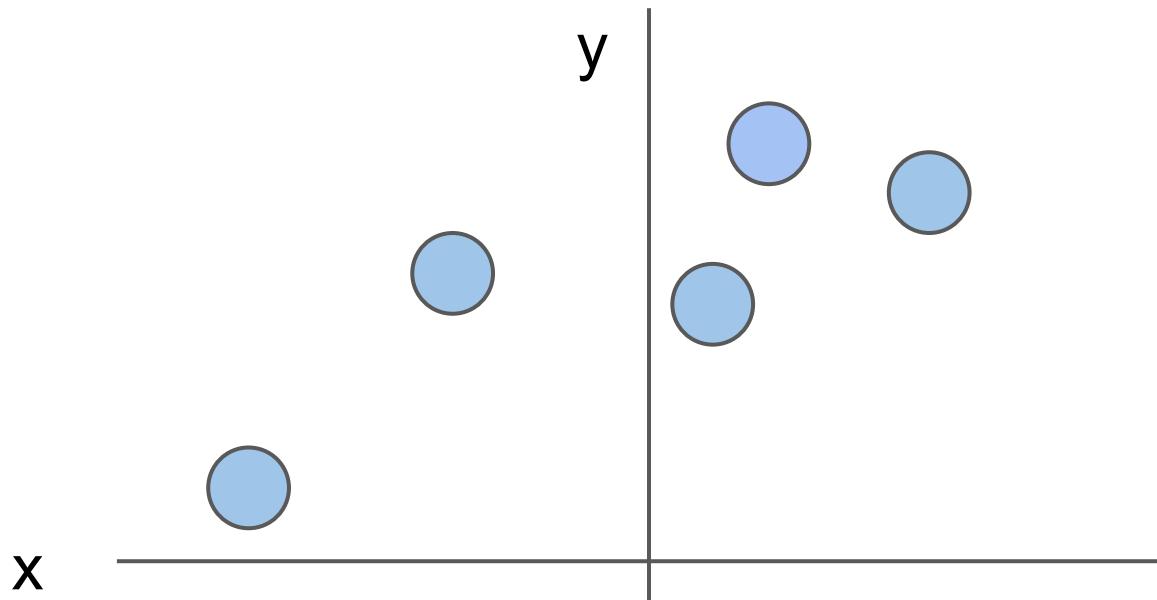
Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda R(W)$$

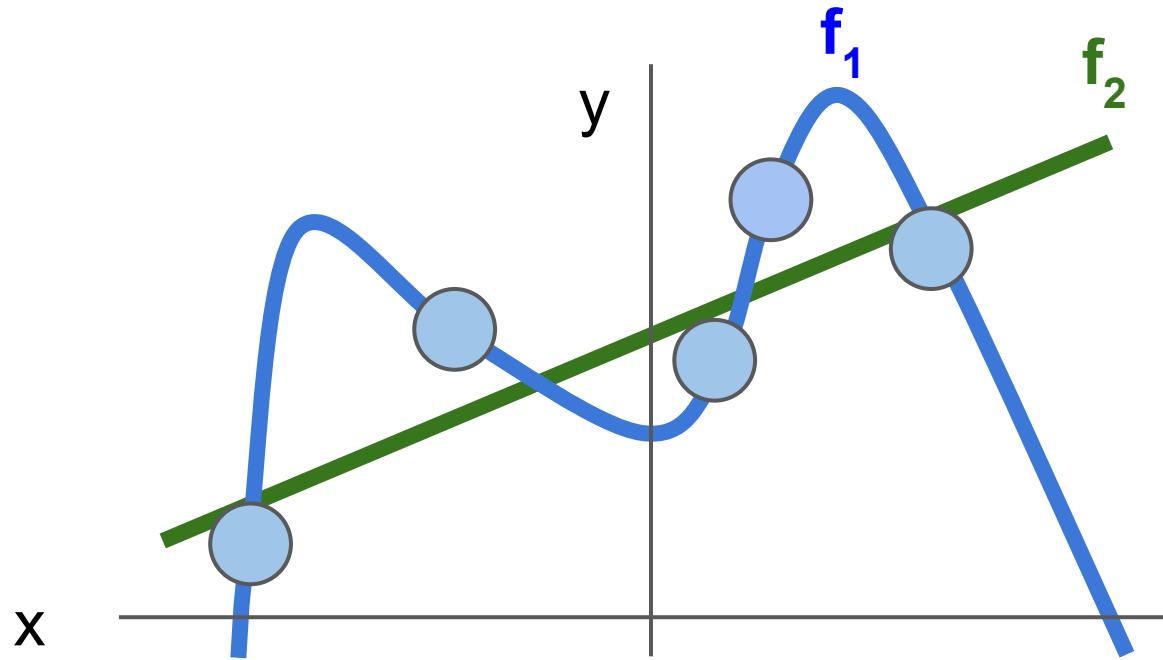

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

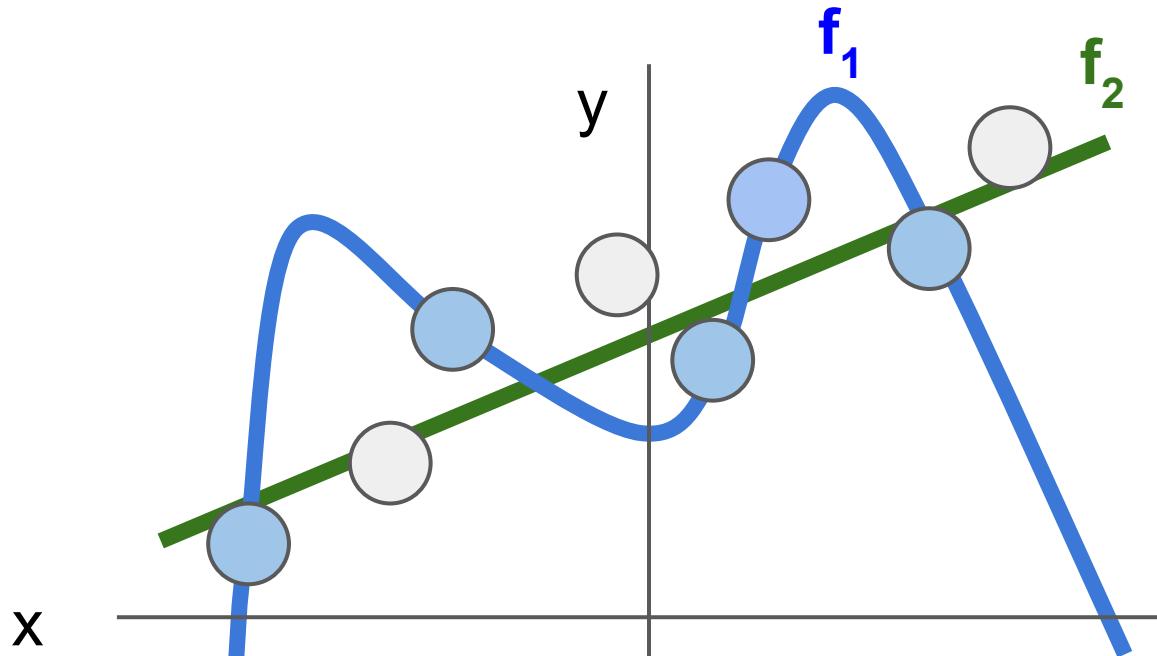
Regularization intuition: toy example training data



Regularization intuition: Prefer Simpler Models



Regularization: Prefer Simpler Models



Regularization pushes against fitting the data
too well so we don't fit noise in the data

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda R(W)$$


Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

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

Regularization

λ = regularization strength
(hyperparameter)

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda R(W)$$



Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

Regularization

λ = regularization strength
(hyperparameter)

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda R(W)$$



Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

Simple examples

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

Regularization

λ = regularization strength
(hyperparameter)

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda R(W)$$



Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

Simple examples

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

More complex:

Dropout

Batch normalization

Stochastic depth, fractional pooling, etc

Regularization

λ = regularization strength
(hyperparameter)

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

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

Why regularize?

- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature

Regularization: Expressing Preferences

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

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

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

L2 Regularization

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

Which of w1 or w2 will
the L2 regularizer prefer?

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

Regularization: Expressing Preferences

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

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

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

L2 Regularization

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

Which of w1 or w2 will
the L2 regularizer prefer?

L2 regularization likes to
“spread out” the weights

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

Regularization: Expressing Preferences

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

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

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

L2 Regularization

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

Which of w1 or w2 will
the L2 regularizer prefer?

L2 regularization likes to
“spread out” the weights

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

Which one would L1
regularization prefer?

Softmax classifier

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



cat	3.2
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



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

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

Softmax
Function

cat	3.2
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

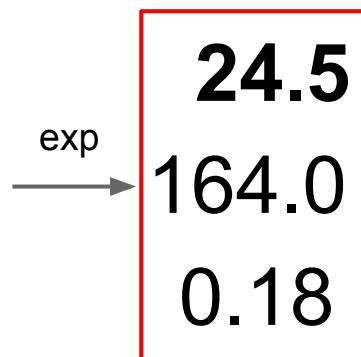
$$s = f(x_i; W)$$

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

Softmax
Function

Probabilities
must be ≥ 0

cat	3.2
car	5.1
frog	-1.7



unnormalized
probabilities

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



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

Probabilities
must be ≥ 0

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

Softmax
Function

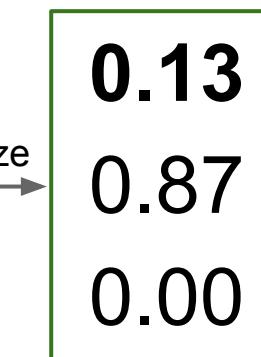
cat	3.2
car	5.1
frog	-1.7

exp



unnormalized
probabilities

normalize



probabilities

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



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

Probabilities
must be ≥ 0

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

Softmax
Function

cat
car
frog

3.2
5.1
-1.7

Unnormalized
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized
probabilities

normalize

0.13
0.87
0.00

probabilities

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



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

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

Probabilities
must be ≥ 0

Probabilities
must sum to 1

cat
car
frog

3.2
5.1
-1.7

Unnormalized
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized
probabilities

normalize

0.13
0.87
0.00

probabilities

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

$$\rightarrow L_i = -\log(0.13) \\ = 2.04$$

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



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

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

Softmax Function

Probabilities
must be ≥ 0

Probabilities
must sum to 1

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

probabilities

Unnormalized
log-probabilities / logits

unnormalized
probabilities

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

$$\rightarrow L_i = -\log(0.13) \\ = 2.04$$

Maximum Likelihood Estimation
Choose weights to maximize the likelihood of the observed data
(See CS 229 for details)

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



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

Probabilities
must be ≥ 0

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

Softmax
Function

cat
car
frog

3.2
5.1
-1.7

Unnormalized
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized
probabilities

normalize

0.13
0.87
0.00

probabilities

compare

1.00
0.00
0.00

Correct
probs

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



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

Probabilities
must be ≥ 0

cat
car
frog

3.2
5.1
-1.7

Unnormalized
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized
probabilities

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

Softmax
Function

Probabilities
must sum to 1

0.13
0.87
0.00

probabilities

normalize

compare

Kullback–Leibler
divergence

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

1.00
0.00
0.00

0.00
0.00

Correct
probs

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

Probabilities
must be ≥ 0

cat
car
frog

3.2
5.1
-1.7

Unnormalized
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized
probabilities

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

Softmax
Function

Probabilities
must sum to 1

0.13
0.87
0.00

probabilities

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

compare

1.00

0.00

0.00

Cross Entropy

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

Correct
probs

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



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

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

Softmax
Function

Maximize probability of correct class

cat	3.2
car	5.1
frog	-1.7

Putting it all together:

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

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

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax
Function

Maximize probability of correct class

Putting it all together:

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

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

cat	3.2
car	5.1
frog	-1.7

Q1: What is the min/max possible softmax loss L_i ?

Q2: At initialization all s_j will be approximately equal; what is the softmax loss L_i , assuming C classes?

Softmax Classifier (Multinomial Logistic Regression)



Want to interpret raw classifier scores as **probabilities**

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

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

Softmax
Function

Maximize probability of correct class

Putting it all together:

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

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

cat	3.2
car	5.1
frog	-1.7

Q2: At initialization all s will be approximately equal; what is the loss?
A: $-\log(1/C) = \log(C)$,
If $C = 10$, then $L_i = \log(10) \approx 2.3$

Softmax vs. 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
22
-44
56

x_i

y_i 2

0.0
0.2
-0.3

$+$

b

-2.85
0.86
0.28

hinge loss (SVM)

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

cross-entropy loss (Softmax)

-2.85
0.86
0.28

\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: What is the **softmax loss** and the **SVM** loss?

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:

[20, -2, 3]

[20, 9, 9]

[20, -100, -100]

and $y_i = 0$

Q: What is the **softmax loss** and the **SVM loss if I double the correct class score from 10 -> 20?**

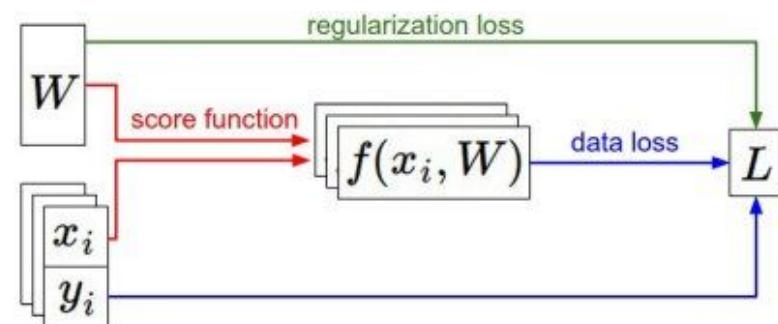
Recap

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

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

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

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

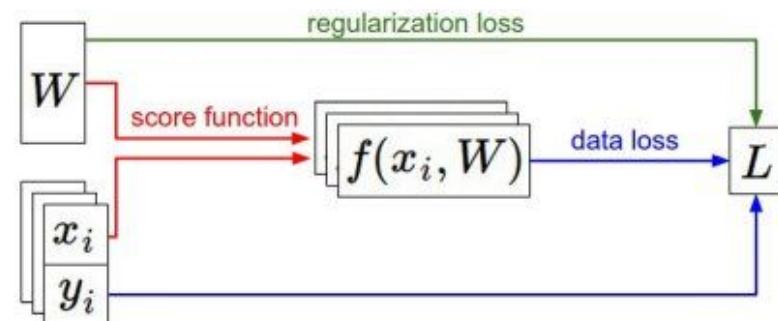


Recap

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 ~99.7%)

Strategy #2: Follow the slope



Strategy #2: Follow the slope

In 1-dimension, the derivative of a function:

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

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

The slope in any direction is the **dot product** of the direction with the gradient
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:	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	[?, ?, ?, ?, ?, ?, ?, ?, ?,...]

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,...] loss 1.25347	[0.34, -1.11 + 0.0001 , 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...] loss 1.25353	[-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,
?,
?]

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

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

?,...]

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 + 0.0001 , 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,
?,
0,...]

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

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

?,...

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,
0,
?,
?]

Numeric Gradient

- Slow! Need to loop over all dimensions
- Approximate

[,...]

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:

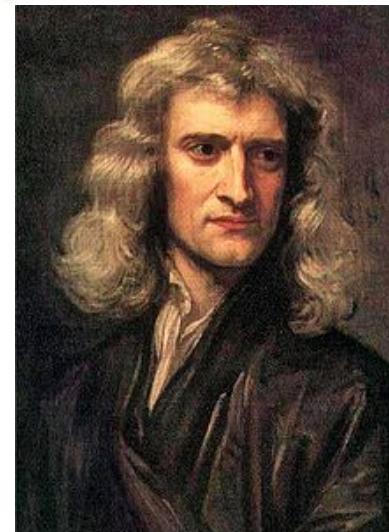
$$L = \frac{1}{N} \sum_{i=1}^N L_i + \sum_k W_k^2$$

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

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

want $\nabla_W L$

Use calculus to compute an analytic gradient



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



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

current W:

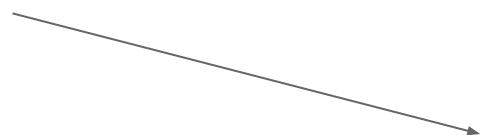
[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

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

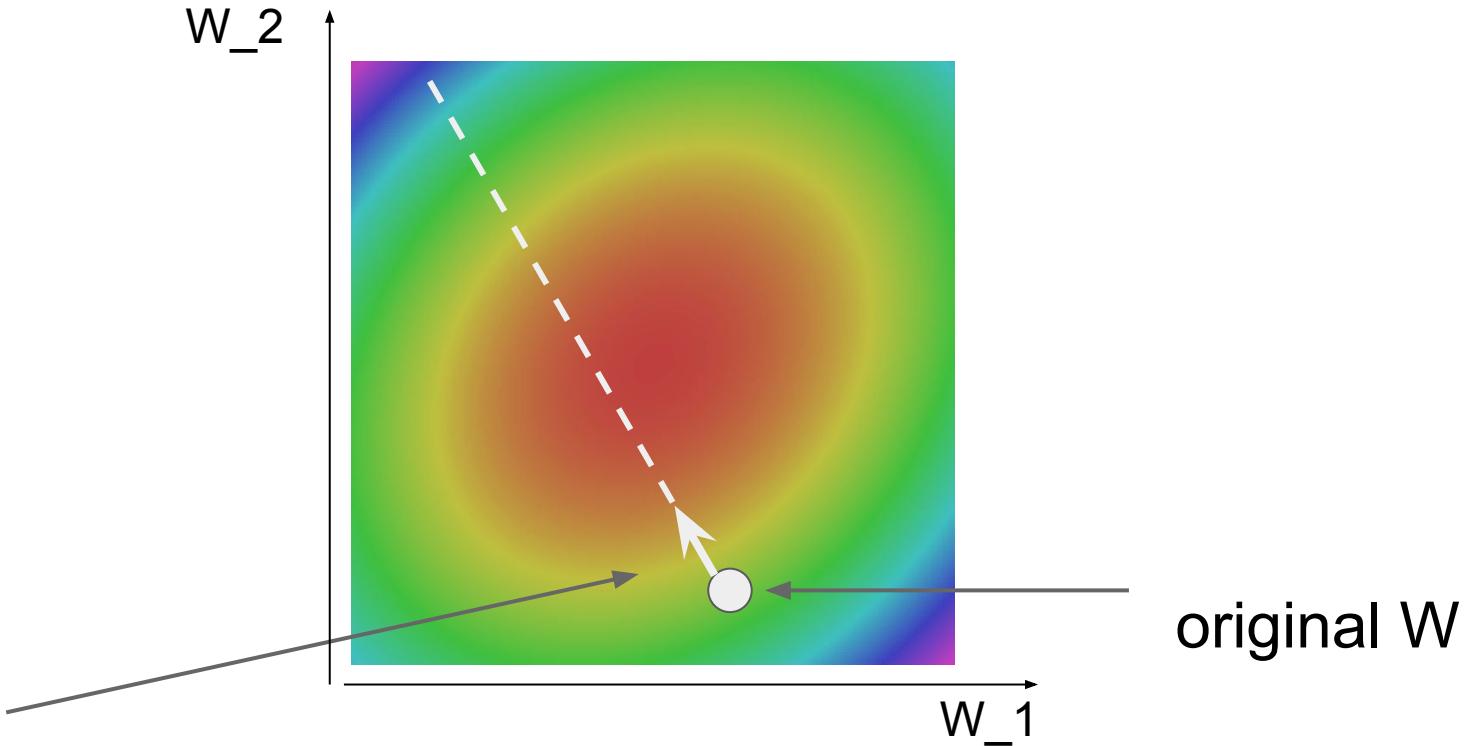
=>

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

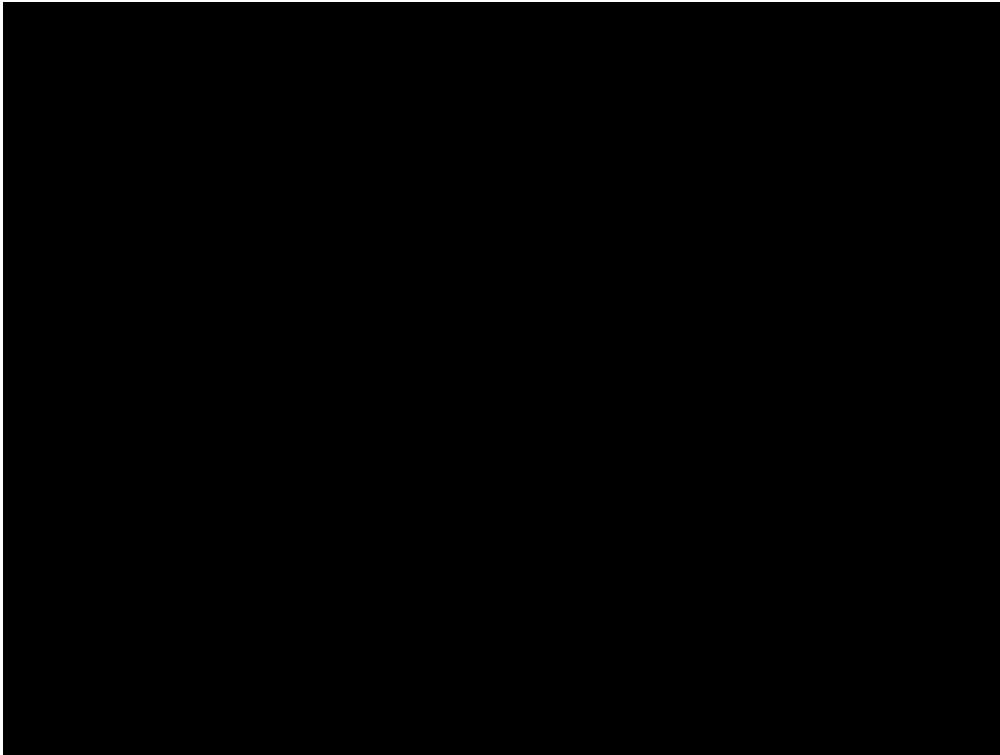
Gradient Descent

```
# 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)

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

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

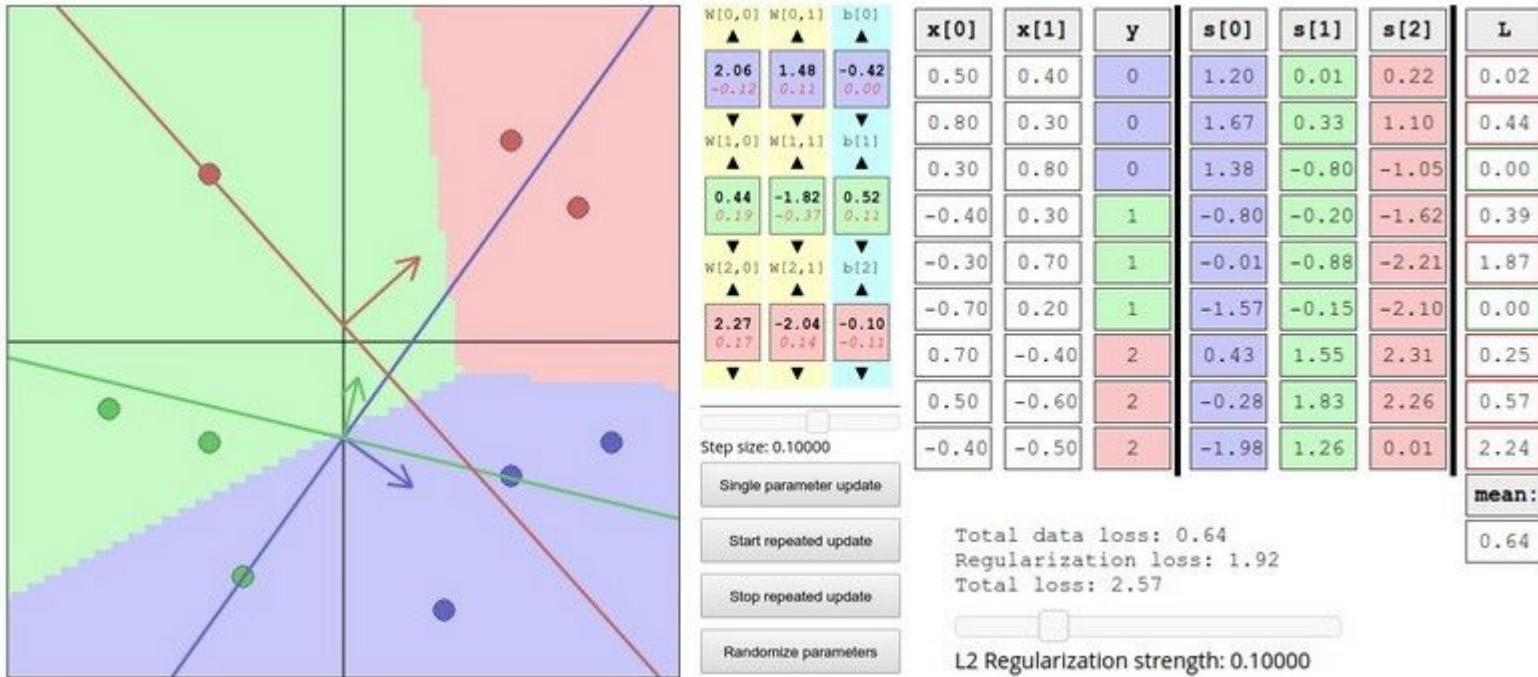
Full sum expensive
when N is large!

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



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

Next time:

Introduction to neural networks

Backpropagation