Lecture 3: Loss Functions and Optimization
Administrative: Assignment 1

Released last week, due **Wed 4/22 at 11:59pm**
Administrative: Project proposal

Due **Wed 4/27**

TA expertise listed on piazza

There is a Piazza thread to find teammates
Administrative: Midterm

24-Hours open notes exam

Combination of True/False, Multiple Choice, Short Answer, Coding

Will be released May 12th, specific time TBD

Due May 13th, 24 hours from release time

The exam should take around 3-hours to finish
Administrative: Piazza

Please make sure to check and read all pinned piazza posts.
Image Classification: A core task in Computer Vision

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

cat

do
t

bird

deer

truck
Recall from last time: Challenges of recognition

- Viewpoint
- Illumination
- Deformation
- Occlusion
- Clutter
- Intraclass Variation
Recall from last time: data-driven approach, kNN

airplane  automobile  
bird    cat    deer    
dog    frog    horse    
ship    truck

train  test

train  validation  test
Recall from last time: Linear Classifier

\[ f(x, W) = Wx + b \]

**Algebraic Viewpoint**

\[ f(x, W) = Wx \]

**Visual Viewpoint**

One template per class

**Geometric Viewpoint**

Hyperplanes cutting up space

Class 1:
1 \leq L2 norm \leq 2

Class 2:
Everything else
Interpreting a Linear Classifier: Visual Viewpoint

Fei-Fei Li, Ranjay Krishna, Danfei Xu  
Lecture 3 - 10  
April 14, 2020
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

**Algebraic Viewpoint**

\[ f(x, W) = Wx \]

**Visual Viewpoint**

Input image

\[
\begin{pmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
56 \\
231 \\
24 \\
2 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
56 \\
231 \\
24 \\
2 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
-96.8 \\
437.9 \\
61.95 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
1.1 \\
3.2 \\
-1.2 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
-96.8 \\
437.9 \\
61.95 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & .25 \\
0.2 & -0.3 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
1.1 \\
3.2 \\
-1.2 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
-96.8 \\
437.9 \\
61.95 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & .25 \\
0.2 & -0.3 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
1.1 \\
3.2 \\
-1.2 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
-96.8 \\
437.9 \\
61.95 \\
\end{pmatrix}
\]
Interpreting a Linear Classifier: Geometric Viewpoint

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

Array of $32 \times 32 \times 3$ numbers (3072 numbers total)

Fei-Fei Li, Ranjay Krishna, Danfei Xu
Recall from last time: Linear Classifier

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)*

<table>
<thead>
<tr>
<th></th>
<th>airplane</th>
<th>automobile</th>
<th>bird</th>
<th>cat</th>
<th>deer</th>
<th>dog</th>
<th>frog</th>
<th>horse</th>
<th>ship</th>
<th>truck</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-3.45</td>
<td>-0.51</td>
<td>3.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-8.87</td>
<td>6.04</td>
<td>4.64</td>
<td>2.9</td>
<td>-4.22</td>
<td>5.1</td>
<td>4.49</td>
<td>1.06</td>
<td>5.55</td>
<td>6.14</td>
</tr>
</tbody>
</table>

TODO:

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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>3.2</th>
<th>1.3</th>
<th>2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
<td></td>
</tr>
</tbody>
</table>
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

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 an 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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

**Multiclass SVM loss:**

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ 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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>f(x)</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>s_{y_i}</td>
<td>-1.7</td>
<td>5.1</td>
<td>3.2</td>
</tr>
<tr>
<td>s_j</td>
<td>1.3</td>
<td>2.0</td>
<td>2.5</td>
</tr>
<tr>
<td>s_j - s_{y_i} + 1</td>
<td>4.9</td>
<td>3.1</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Interpreting Multiclass SVM loss:

The loss function $L_i$ is defined as:

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

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>score</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>score</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>score</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>score</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Interpreting Multiclass SVM loss:

Given an example where $x$ is the image and $y$ is the (integer) label, and using the shorthand for the scores vector: the SVM loss has the form:

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
<td></td>
</tr>
</tbody>
</table>

**Multiclass SVM loss:**

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ 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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>frog</th>
<th>car</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
<td></td>
</tr>
</tbody>
</table>

**Multiclass SVM loss:**

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$ the SVM loss has the form:

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

$$= \max(0, 5.1 - 3.2 + 1) + \max(0, -1.7 - 3.2 + 1)$$

$$= \max(0, 2.9) + \max(0, -3.9)$$

$$= 2.9 + 0$$

$$= 2.9$$
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>frog</th>
<th>car</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>2</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>3</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

**Multiclass SVM loss:**

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$, the SVM loss has the form:

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

$$= \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1)$$

$$= \max(0, -2.6) + \max(0, -1.9)$$

$$= 0 + 0$$

$$= 0$$
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>Cat</th>
<th>Car</th>
<th>Frog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>Cat</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>Frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Losses: 2.9 0 12.9

### Multiclass SVM loss:

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

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

the SVM loss has the form:

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

\[
= \max(0, 2.2 - (-3.1) + 1) + \max(0, 2.5 - (-3.1) + 1)
\]

\[
= \max(0, 6.3) + \max(0, 6.6)
\]

\[
= 6.3 + 6.6 = 12.9
\]
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

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:

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

$L = (2.9 + 0 + 12.9)/3 = 5.27$
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = W x$ are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1.3</td>
</tr>
<tr>
<td>car</td>
<td>4.9</td>
</tr>
<tr>
<td>frog</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Losses: 0

**Multiclass SVM loss:**

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

**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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>score</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>score</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

**Multiclass SVM loss:**

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ 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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>1.3</th>
<th>2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
<tr>
<td></td>
<td>2.9</td>
<td>0</td>
<td>12.9</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th></th>
<th>Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

Multiclass SVM loss:

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ 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)
    margins = np.maximum(0, scores - scores[y] + 1)
    margins[y] = 0
    loss_i = np.sum(margins)
    return loss_i
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:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>3.2</th>
<th>1.3</th>
<th>2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
<td></td>
</tr>
</tbody>
</table>

Losses: 2.9

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

Before:

\[
= \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1)
= \max(0, -2.6) + \max(0, -1.9)
= 0 + 0
= 0
\]

With \( W \) twice as large:

\[
= \max(0, 2.6 - 9.8 + 1) + \max(0, 4.0 - 9.8 + 1)
= \max(0, -6.2) + \max(0, -4.8)
= 0 + 0
= 0
\]
\[ f(x, W) = Wx \]

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

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

No! 2\( W \) is also has \( L = 0 \! \)!
How do we choose between \( W \) and 2\( W \)?
Regularization

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

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

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

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

**Regularization:** 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

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

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

**Regularization**: 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

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

- **Data loss**: Model predictions should match training data
- **Regularization**: Prevent the model from doing too well on training data

\( \lambda \) = regularization strength (hyperparameter)
Regularization

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

Data loss: Model predictions should match training data

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

\[ \lambda = \text{regularization strength (hyperparameter)} \]

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

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

\( \lambda = \) regularization strength (hyperparameter)

**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

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

\( \lambda \) = regularization strength (hyperparameter)

**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

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

\[ w_1^T x = w_2^T x = 1 \]
Regularization: Expressing Preferences

L2 Regularization

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

L2 regularization likes to “spread out” the weights

\[ w_1^T x = w_2^T x = 1 \]  

\[ x = [1, 1, 1, 1] \]
\[ w_1 = [1, 0, 0, 0] \]
\[ w_2 = [0.25, 0.25, 0.25, 0.25] \]
Regularization: Expressing Preferences

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

\[ w_1^T x = w_2^T x = 1 \]

L2 Regularization

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

L2 regularization likes to “spread out” the weights

Which one would L1 regularization prefer?
Softmax Classifier (Multinomial Logistic Regression)

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Class</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Class</th>
<th>Score</th>
<th>Unnormalized Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>24.5</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>164.0</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Fei-Fei Li, Ranjay Krishna, Danfei Xu
**Softmax Classifier** (Multinomial Logistic Regression)

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

\[ s = f(x_i; W) \]

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

Probabilities must be \( \geq 0 \)

Probabilities must sum to 1

<table>
<thead>
<tr>
<th></th>
<th>Cat</th>
<th>Car</th>
<th>Frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>exp</td>
<td>24.5</td>
<td>164.0</td>
<td>0.18</td>
</tr>
<tr>
<td>normalize</td>
<td>0.13</td>
<td>0.87</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**unnormalized probabilities**
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

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

Probabilities must be >= 0

Probabilities must sum to 1

Unnormalized log-probabilities / logits

Unnormalized probabilities

probabilities

cat  3.2  24.5  0.13

frog -1.7  0.18  0.00

car  5.1  164.0  0.87

\[ \exp \]

\[ \text{normalize} \]
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 \( \geq 0 \)

Probabilities must sum to 1

Unnormalized log-probabilities / logits

Unnormalized probabilities

Probabilities

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

\[ L_i = -\log(0.13) \]

\[ = 2.04 \]
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[
s = f(x_i; W)
\]

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

Probabilities must be \( \geq 0 \)

Probabilities must sum to 1

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

\[
\rightarrow L_i = -\log(0.13) = 2.04
\]

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**

\[ s = f(x_i; W) \]

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

**Softmax Function**

- Probabilities must be \( \geq 0 \)
- Probabilities must sum to 1

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

**Objective Function**

Unnormalized log-probabilities / logits

<table>
<thead>
<tr>
<th>cat</th>
<th>3.2</th>
<th>24.5</th>
<th>0.13</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>car</td>
<td>5.1</td>
<td>164.0</td>
<td>0.87</td>
<td>0.00</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>0.18</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**Correct probs**
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

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

Probabilities must be >= 0

Probabilities must sum to 1

Unnormalized log-probabilities / logits

\[ \exp \]

normalize

\[ \text{exp} \]

\[ \text{normalize} \]

Correct probs

Kullback–Leibler divergence

\[ D_{KL}(P||Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)} \]
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 \( \geq 0 \)

Probabilities must sum to 1

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

Cross Entropy

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

Unnormalized log-probabilities / logits

Unnormalized probabilities

Probabilities

Correct probs

cat

3.2

24.5

0.13

1.00

Unnormalized log-probabilities / logits

unnormalized probabilities

probabilities

Correct probs
Softmax Classifier (Multinomial Logistic Regression)

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

\[ s = f(x_i; W) \]

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_{j} e^{s_j}} \]

Maximize probability of correct class

Maximize probability of correct class

Putting it all together:

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

\[ L_i = - \log \left( \frac{e^{s_{y_i}}}{\sum_{j} e^{s_j}} \right) \]

<table>
<thead>
<tr>
<th>Class</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

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

Maximize probability of correct class

Putting it all together:

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

\[ L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}) \]

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

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

Maximize probability of correct class

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

Putting it all together:

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

Q: What is the min/max possible loss \( L_i \)?
A: min 0, max infinity

<table>
<thead>
<tr>
<th>Image</th>
<th>Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat</td>
<td>3.2</td>
</tr>
<tr>
<td>Car</td>
<td>5.1</td>
</tr>
<tr>
<td>Frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

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

Maximize probability of correct class

Putting it all together:

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

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

Q2: At initialization all \( s \) will be approximately equal; what is the loss?
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

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

Maximize probability of correct class

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

Putting it all together:

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

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

\[ W \]

\[
\begin{array}{cccc}
0.01 & -0.05 & 0.1 & 0.05 \\
0.7 & 0.2 & 0.05 & 0.16 \\
0.0 & -0.45 & -0.2 & 0.03 \\
\end{array}
\]

\[
\begin{array}{c}
-15 \\
22 \\
-44 \\
56 \\
\end{array}
\]

\[
\begin{array}{c}
0 \\
0.2 \\
-0.3 \\
\end{array}
\]

\[ b \]

\[ x_i \]

\[
\begin{array}{c}
0.058 \\
2.36 \\
0.28 \\
1.32 \\
\end{array}
\]

\[ exp \]

\[
\begin{array}{c}
0.86 \\
0.28 \\
\end{array}
\]

\[ normalize (to sum to one) \]

\[
\begin{array}{c}
0.016 \\
0.631 \\
0.353 \\
\end{array}
\]

\[ y_i \]

2

hinge loss (SVM)

\[
\max(0, -2.85 - 0.28 + 1) + \max(0, 0.86 - 0.28 + 1)
\]

= 1.58

cross-entropy loss (Softmax)

\[
-\log(0.353) = 0.452\]
Softmax vs. SVM

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

\[
\begin{align*}
[10, -2, 3] \\
[10, 9, 9] \\
[10, -100, -100]
\end{align*}
\]

and \( y_i = 0 \)

Q: What is the softmax loss and the SVM loss?
Softmax vs. SVM

\[ L_i = - \log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}) \]

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

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 if I double the correct class score from 10 -> 20?
Recap

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

  \[
  L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \quad \text{Softmax}
  \]

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

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

- We have some dataset of \((x, y)\)
- We have a score function: \(s = f(x; W) = Wx\)
- We have a loss function:

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

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

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W) \quad \text{Full loss}
\]
Optimization
Strategy #1: A first very bad idea solution: Random search

```python
# 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 9.270940, 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)
```
Let's see how well this works on the test set...

```python
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555

15.5% accuracy! not bad!
(SOTA is ~99.3%)
```
Strategy #2: Follow the slope
Strategy #2: Follow the slope

In 1-dimension, the derivative of a function:

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

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:

<table>
<thead>
<tr>
<th>current W:</th>
<th>( W + h ) (first dim):</th>
<th>gradient dW:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34,</td>
<td>[0.34 + 0.0001,</td>
<td>[?,</td>
</tr>
<tr>
<td>-1.11,</td>
<td>-1.11,</td>
<td>?,</td>
</tr>
<tr>
<td>0.78,</td>
<td>0.78,</td>
<td>?,</td>
</tr>
<tr>
<td>0.12,</td>
<td>0.12,</td>
<td>?,</td>
</tr>
<tr>
<td>0.55,</td>
<td>0.55,</td>
<td>?,</td>
</tr>
<tr>
<td>2.81,</td>
<td>2.81,</td>
<td>?,</td>
</tr>
<tr>
<td>-3.1,</td>
<td>-3.1,</td>
<td>?,</td>
</tr>
<tr>
<td>-1.5,</td>
<td>-1.5,</td>
<td>?,</td>
</tr>
<tr>
<td>0.33,...]</td>
<td>0.33,...]</td>
<td>?,...</td>
</tr>
</tbody>
</table>

loss 1.25347

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

$W + \Delta h$ (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 $dW$:  
\[ [-2.5, ?, ?, ?, ?, ?, ?, 1.25322 - 1.25347]/0.0001 = -2.5 \]

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$
<table>
<thead>
<tr>
<th>current W:</th>
<th>( W + h ) (second dim):</th>
<th>gradient ( dW ):</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33, \ldots]</td>
<td>[0.34, -1.11 + 0.0001, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33, \ldots]</td>
<td>([-2.5, ?, ?, ?, ?, ?, ?, \ldots])</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25353</td>
<td></td>
</tr>
</tbody>
</table>
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, |
| ?, |
| ?, |
| ?, |
| ?, | ] 

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

\[
\frac{df(x)}{dx} = \lim_{{h \to 0}} \frac{f(x + h) - f(x)}{h}
\]
<table>
<thead>
<tr>
<th>current W:</th>
<th>$W + h$ (third dim):</th>
<th>gradient $dW$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34,</td>
<td>[0.34,</td>
<td>[-2.5,</td>
</tr>
<tr>
<td>-1.11,</td>
<td>-1.11,</td>
<td>0.6,</td>
</tr>
<tr>
<td>0.78,</td>
<td>0.78 + 0.0001,</td>
<td>?,</td>
</tr>
<tr>
<td>0.12,</td>
<td>0.12,</td>
<td>?,</td>
</tr>
<tr>
<td>0.55,</td>
<td>0.55,</td>
<td>?,</td>
</tr>
<tr>
<td>2.81,</td>
<td>2.81,</td>
<td>?,</td>
</tr>
<tr>
<td>-3.1,</td>
<td>-3.1,</td>
<td>?,</td>
</tr>
<tr>
<td>-1.5,</td>
<td>-1.5,</td>
<td>?,</td>
</tr>
<tr>
<td>0.33,...]</td>
<td>0.33,...]</td>
<td>?,</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25347</td>
<td>...</td>
</tr>
<tr>
<td>current $W$:</td>
<td>$W + h$ (third dim):</td>
<td>gradient $dW$:</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33, ...]</td>
<td>[0.34, -1.11, 0.78 + 0.0001, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33, ...]</td>
<td>[-2.5, 0.6, 0, ?, ?, ?, ?]</td>
</tr>
</tbody>
</table>
| loss 1.25347          | loss 1.25347                  | \[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]
|                      |                               | \[
\frac{(1.25347 - 1.25347)/0.0001}{0.0001} = 0
\] |
<table>
<thead>
<tr>
<th>current W:</th>
<th>W + h (third dim):</th>
<th>gradient dW:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34, -1.11, 0.78 + 0.0001, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[-2.5, 0.6, 0, ?, 0, ?, 0, ?,... ]</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25347</td>
<td>Numeric Gradient</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Slow! Need to loop over all dimensions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Approximate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>?,...]</td>
</tr>
</tbody>
</table>
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
current $W$: [0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, -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

```python
# Vanilla Gradient Descent

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

original $W$

$W_1$

$W_2$
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

```python
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