Lecture 3:
Loss functions and Optimization
Administrative

A1 is due Jan 20 (Wednesday). ~9 days left
Warning: Jan 18 (Monday) is Holiday (no class/office hours)
Recall from last time… Challenges in Visual Recognition

- Camera pose
- Illumination
- Deformation
- Occlusion
- Background clutter
- Intraclass variation
Recall from last time… data-driven approach, kNN

The data

NN classifier

5-NN classifier

Cross-validation on k

k

0.24
0.25
0.26
0.27
0.28
0.29
0.30
0.31
0.32

Lecture 3 - 11 Jan 2016
Recall from last time… Linear classifier

\[ f(x, W) \]

$[32 \times 32 \times 3]$ array of numbers 0...1 (3072 numbers total)

Input image

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>-0.5</td>
<td>0.1</td>
<td>2.0</td>
</tr>
<tr>
<td>1.5</td>
<td>1.3</td>
<td>2.1</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.25</td>
<td>0.2</td>
<td>-0.3</td>
</tr>
</tbody>
</table>
```

$W$

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>231</td>
<td>24</td>
<td>2</td>
</tr>
</tbody>
</table>
```

$b$

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>-96.8</td>
<td>437.9</td>
<td>61.95</td>
</tr>
</tbody>
</table>
```

Parameters

10 numbers, indicating class scores
Recall from last time… Going forward: Loss function/Optimization

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

TODO:

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

2. Come up with a way of efficiently finding the parameters that minimize the loss function. (optimization)
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</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>
</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>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</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>
</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></td>
<td>3.2</td>
<td>-1.7</td>
<td>5.1</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>2.0</td>
<td>4.9</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>2.5</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:

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

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

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

$$= 2.9 + 0$$

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

\[
\begin{array}{ccc}
\text{cat} & 3.2 & 1.3 & 2.2 \\
\text{car} & 5.1 & 4.9 & 2.5 \\
\text{frog} & -1.7 & 2.0 & -3.1 \\
\end{array}
\]

**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:

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

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

\[
= \max(0, -2.6) + \max(0, -1.9)
\]

\[
= 0 + 0
\]

\[
= 0
\]

Losses:

\[
\begin{array}{ccc}
\text{cat} & 3.2 & 1.3 & 2.2 \\
\text{car} & 5.1 & 4.9 & 2.5 \\
\text{frog} & -1.7 & 2.0 & -3.1 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Losses:} \\
\text{2.9} \\
\text{0}
\end{array}
\]
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></th>
<th></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>
<tr>
<td>Losses:</td>
<td>2.9</td>
<td>0</td>
<td>10.9</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:

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

$$= \max(0, 2.2 - (-3.1) + 1) + \max(0, 2.5 - (-3.1) + 1)$$
$$= \max(0, 5.3) + \max(0, 5.6)$$
$$= 5.3 + 5.6$$
$$= 10.9$$
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></th>
<th></th>
<th>car</th>
<th></th>
<th></th>
<th></th>
<th>frog</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
<td></td>
</tr>
</tbody>
</table>

**Losses:**

- 2.9
- 0
- 10.9

**Multiclass SVM loss:**

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

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

the SVM loss has the form:

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

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

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

\[
L = \frac{(2.9 + 0 + 10.9)}{3} = 4.6
\]
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>

Losses: 2.9 0 10.9

**Multiclass SVM loss:**

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

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

the SVM loss has the form:

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

**Q:** what if the sum was instead over all classes? (including $j = y_i$)
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

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

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

**Q2:** what if we used a mean instead of a sum here?
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>3.2</th>
<th>1.3</th>
<th>2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td></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>
</tbody>
</table>

**Losses:** 2.9 0 10.9

**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:

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

**Q3: what if we used**

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th></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>

**Losses:** 2.9  0  10.9

**Multiclass SVM loss:**

Given an example $\left(x_i, y_i\right)$ where $x_i$ is the image and where $y_i$ is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Q4: what is the min/max possible loss?
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

<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>

Losses: 2.9 0 10.9

**Multiclass SVM loss:**

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

the SVM loss has the form:

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

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

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

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

\[ L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) \]
There is a bug with the loss:

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

\[ L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) \]
There is a bug with the loss:

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

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

E.g. Suppose that we found a \( W \) such that \( L = 0 \). Is this \( W \) unique?
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

\[
\begin{array}{c|ccc}
\text{cat} & 3.2 & 1.3 & 2.2 \\
\text{car} & 5.1 & 4.9 & 2.5 \\
\text{frog} & -1.7 & 2.0 & -3.1 \\
\end{array}
\]

Losses: 2.9 0 -3.1

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

**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:**
\[
= \max(0, 2.6 - 9.8 + 1) + \max(0, 4.0 - 9.8 + 1)
\]
\[
= \max(0, -6.2) + \max(0, -4.8)
\]
\[
= 0 + 0
\]
\[
= 0
\]
Weight Regularization

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

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

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)
L2 regularization: motivation

\[ x = [1, 1, 1, 1] \]

\[ w_1 = [1, 0, 0, 0] \]

\[ w_2 = [0.25, 0.25, 0.25, 0.25] \]

\[ w_1^T x = w_2^T x = 1 \]
Softmax Classifier (Multinomial Logistic Regression)

cat  3.2

frog  -1.7
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

cat  3.2

car  5.1

frog -1.7
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

where

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

cat  3.2

car  5.1

tfrog -1.7
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

where

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

cat  3.2

car  5.1

gord  -1.7

Softmax function
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

where \( s = f(x_i; W) \)

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

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

cat 3.2

car 5.1

frog -1.7
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

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

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

in summary:

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

cat 3.2

car 5.1

frog -1.7
Softmax Classifier (Multinomial Logistic Regression)

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Unnormalized Log Probability</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>

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

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

<table>
<thead>
<tr>
<th></th>
<th>unnormalized log probabilities</th>
<th>unnormalized probabilities</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>
Softmax Classifier (Multinomial Logistic Regression)

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

<table>
<thead>
<tr>
<th></th>
<th>unnormalized log probabilities</th>
<th>exp</th>
<th>normalize</th>
<th>probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>24.5</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>164.0</td>
<td>0.87</td>
<td></td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>0.18</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>
Softmax Classifier (Multinomial Logistic Regression)

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

<table>
<thead>
<tr>
<th>Class</th>
<th>Log Probability</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>24.5</td>
<td>0.13</td>
</tr>
<tr>
<td>car</td>
<td>164.0</td>
<td>0.87</td>
</tr>
<tr>
<td>frog</td>
<td>0.18</td>
<td>0.00</td>
</tr>
</tbody>
</table>

\(L_i = -\log(0.13) = 0.89\)
### Softmax Classifier (Multinomial Logistic Regression)

The Softmax Classifier is a type of logistic regression model that is used for classification tasks where the output is a probability distribution over multiple classes. It is particularly useful for problems where the classes are not linearly separable.

#### Unnormalized Log Probabilities

<table>
<thead>
<tr>
<th>Class</th>
<th>Unnormalized Log Probability</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>

#### Normalized Probabilities

The unnormalized log probabilities are converted to probabilities by the softmax function:

\[
\text{softmax}(x) = \frac{e^x}{\sum e^y}
\]

<table>
<thead>
<tr>
<th>Class</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>0.13</td>
</tr>
<tr>
<td>car</td>
<td>0.87</td>
</tr>
<tr>
<td>frog</td>
<td>0.00</td>
</tr>
</tbody>
</table>

The loss function for the Softmax Classifier is given by:

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

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

\[
L_i = -\log(0.13) = 0.89
\]
Softmax Classifier (Multinomial Logistic Regression)

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

Unnormalized log probabilities:

- **cat:** 3.2
- **car:** 5.1
- **frog:** -1.7

Unnormalized probabilities:

- **cat:** \( \exp(3.2) = 24.5 \)
- **car:** \( \exp(5.1) = 164.0 \)
- **frog:** \( \exp(-1.7) = 0.18 \)

Probabilities:

- **cat:** \( \frac{24.5}{24.5 + 164.0 + 0.18} = 0.13 \)
- **car:** \( \frac{164.0}{24.5 + 164.0 + 0.18} = 0.87 \)
- **frog:** \( \frac{0.18}{24.5 + 164.0 + 0.18} = 0.00 \)

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

\[ L_i = -\log(0.13) = 0.89 \]
### Hinge Loss (SVM)

\[
\max(0, -2.85 - 0.28 + 1) + \\
\max(0, 0.86 - 0.28 + 1) = 1.58
\]

### Cross-Entropy Loss (Softmax)

- \( \exp(2.36) = 0.016 \)
- \( \frac{0.058}{0.353} = 0.16 \)
- \( \frac{0.28}{0.353} = 0.8 \)
- \( 
\begin{align*}
\text{normalize (to sum to one)} & \rightarrow 0.631 \\
= 0.353 & \rightarrow 0.452
\end{align*}
\)

Matrix multiply + bias offset

\[
\begin{align*}
\begin{bmatrix}
0.01 & -0.05 & 0.1 & 0.05 \\
0.7 & 0.2 & 0.05 & 0.16 \\
0.0 & -0.45 & -0.2 & 0.03
\end{bmatrix}
\begin{bmatrix}
-15 \\
22 \\
-44 \\
56
\end{bmatrix}
& +
\begin{bmatrix}
0.0 \\
0.2 \\
-0.3 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{bmatrix}
y_i \quad 2
\end{bmatrix}
\]
Softmax vs. SVM

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

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

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

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

**assume scores:**

- \([10, -2, 3]\)
- \([10, 9, 9]\)
- \([10, -100, -100]\)

and \(y_i = 0\)

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

http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/
Optimization
Recap

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

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

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

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W) \quad \text{Full loss}
\]
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 9.278940, 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 ~95%)
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).
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:

[?,
?,
?,
?,
?,
?,
?,
?,
?,
?,
?,
?,
?,
?,]
<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>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25322</td>
<td></td>
</tr>
<tr>
<td>current W:</td>
<td>W + h (first 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 + 0.0001, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[\text{-2.5, }？, ？, (\frac{(1.25322 - 1.25347)}{0.0001}) = -2.5 | ? ?, ?,...]</td>
</tr>
</tbody>
</table>

loss 1.25347  
loss 1.25322
<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,...</td>
<td>0.34, -1.11 + 0.0001, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...</td>
<td>[-2.5, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, …]</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25353</td>
<td></td>
</tr>
<tr>
<td>current ( W ):</td>
<td>( W + h ) (second 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 + 0.0001, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33, ...]</td>
<td>[-2.5, 0.6, ?, ?, ?, ?, ?, ...]</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25353</td>
<td></td>
</tr>
</tbody>
</table>

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

\[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]
Evaluation the gradient numerically

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

```python
def eval_numerical_gradient(f, x):
    """
    A naive implementation of numerical gradient of f at x
    - f should be a function that takes a single argument
    - x is the point (numpy array) to evaluate the gradient at
    """
    fx = f(x) # evaluate function value at original point
    grad = np.zeros(x.shape)
    h = 0.00001

    # iterate over all indexes in x
    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:
        # evaluate function at x+h
        ix = it.multi_index
        old_value = x[ix]
        x[ix] = old_value + h # increment by h
        fxh = f(x) # evaluate f(x+h)
        x[ix] = old_value # restore to previous value (very important!)

        # compute the partial derivative
        grad[ix] = (fxh - fx) / h # the slope
        it.iternext() # step to next dimension

    return grad
```
Evaluation the gradient numerically

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

- approximate
- very slow to evaluate

```python
def eval_numerical_gradient(f, x):
    
    # a naive implementation of numerical gradient of f at x
    # f should be a function that takes a single argument
    # x is the point (numpy array) to evaluate the gradient at
    
    fx = f(x)  # evaluate function value at original point
    grad = np.zeros(x.shape)
    h = 0.00001

    # iterate over all indexes in x
    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:

        # evaluate function at x+h
        ix = it.multi_index
        old_value = x[ix]
        x[ix] = old_value + h  # increment by h
        fxh = f(x)  # evaluate f(x + h)
        x[ix] = old_value  # restore to previous value (very important!)

        # compute the partial derivative
        grad[ix] = (fxh - fx) / h  # the slope
        it.iternext()  # step to next dimension

    return grad
```
This is silly. The loss is just a function of $W$:

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

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

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

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

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

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

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

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

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

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

\[
s = f(x; W) = Wx
\]

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

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

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

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

$$\nabla_W L = \ldots$$
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 \( dW \):

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

d\(W = ... \) 
(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 \( W \)
Mini-batch Gradient Descent

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

```python
# Vanilla Minibatch Gradient Descent

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

Common mini-batch sizes are 32/64/128 examples
e.g. Krizhevsky ILSVRC ConvNet used 256 examples
Example of optimization progress while training a neural network.

(Loss over mini-batches goes down over time.)
The effects of step size (or “learning rate”)
Mini-batch Gradient Descent

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

```python
# Vanilla Minibatch Gradient Descent

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

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

we will look at more fancy update formulas (momentum, Adagrad, RMSProp, Adam, …)
The effects of different update form formulas
Aside: Image Features
Example: Color (Hue) Histogram
Example: HOG/SIFT features

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

(image from vlfeat.org)
Example: HOG/SIFT features

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

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

(image from vlfeat.org)
Example: Bag of Words

- Visual word vectors
- Learn k-means centroids
- "Vocabulary" of visual words
- Histogram of visual words

144

E.g. 1000 centroids

1000-d vector
Feature Extraction

[32x32x3]

vector describing various image statistics

10 numbers, indicating class scores

training

[32x32x3]

10 numbers, indicating class scores

training
Next class:

Becoming a backprop ninja
and
Neural Networks (part 1)