High Level Computer Vision

Image Classification, Linear Classifier & Losses
@ April 17, 2019

ver-april18: corrections: p89 (equation), replaced f(x) with f(W) and x with W on several slides: 100, 103, 104, 105, 111

Bernt Schiele & Mario Fritz

www.mpi-inf.mpg.de/hlcv/

Max Planck Institute for Informatics & Saarland University, Saarland Informatics Campus Saarbrücken
Overview Lecture (preliminary)

- "Classic" Computer Vision Tasks, CNNs e.g. for
  - Image Classification
  - Object Detection
  - Semantic Segmentation

- Recurrent Neural Networks, e.g. for
  - Image Captioning
  - Visual Question Answering
  - Visual Touring Test

- How to deal with Less Supervision

- Generative Adversarial Networks e.g. for data generation

- Memory Networks e.g. for VQA
Overview Today’s Lecture

• Image Classification
  ‣ using data-driven approaches (machine learning)
  ‣ K-nearest neighbor classifier
  ‣ linear classification (parametric approach)
  ‣ loss functions and regularization
  ‣ softmax classifier
  ‣ optimization via gradient descent

• Image Features vs. ConvNets

• Slide credit:
  ‣ today’s slides taken from Fei-Fei Li, Justin Johnson, Serena Yeung @ Stanford
Image Classification: a core task in computer vision

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

\[\text{cat}\]
Image Classification

The Problem: Semantic Gap

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

This image by Nihita is licensed under CC BY 2.0
Image Classification

Challenges: Viewpoint variation

All pixels change when the camera moves!
Image Classification

Challenges: Illumination
Image Classification

**Challenges:** Deformation
Image Classification

Challenges: Occlusion
Image Classification

**Challenges:** Background Clutter
Image Classification

Challenges: Intra-class variation
Image Classification

An image classifier

```python
def classify_image(image):
    # Some magic here?
    return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.
Image Classification

Attempts have been made

John Canny, "A Computational Approach to Edge Detection", IEEE TPAMI 1986
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

Example training set

```python
def train(images, labels):
    # Machine learning!
    return model

def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```
First Classifier: Nearest Neighbor

```python
def train(images, labels):
    # Machine learning!
    return model

def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```

- Memorize all data and labels
- Predict the label of the most similar training image
Example Dataset: CIFAR10

10 classes
50,000 training images
10,000 testing images

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Example Dataset: CIFAR10

10 classes
50,000 training images
10,000 testing images

Test images and nearest neighbors

First Classifier: Nearest Neighbor Classifier

Distance Metric to compare images

L1 distance:

\[ d_1(I_1, I_2) = \sum_p |I_{1p} - I_{2p}| \]

<table>
<thead>
<tr>
<th>test image</th>
<th>training image</th>
<th>pixel-wise absolute value differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>56 32 10 18</td>
<td>10 20 24 17</td>
<td>46 12 14 1</td>
</tr>
<tr>
<td>90 23 128 133</td>
<td>8 10 89 100</td>
<td>82 13 39 33</td>
</tr>
<tr>
<td>24 26 178 200</td>
<td>12 16 178 170</td>
<td>12 10 0 30</td>
</tr>
<tr>
<td>2 0 255 220</td>
<td>4 32 233 112</td>
<td>2 32 22 108</td>
</tr>
</tbody>
</table>

\[ \text{add} \rightarrow 456 \]
First Classifier: Nearest Neighbor Classifier

What does this look like?
K-Nearest Neighbor Classifier

Instead of copying label from nearest neighbor, take **majority vote** from K closest points
K-Nearest Neighbor Classifier

What does this look like?
K-Nearest Neighbor Classifier

What does this look like?
K-Nearest Neighbor Classifier: Distance Metric

L1 (Manhattan) distance

\[ d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p| \]

L2 (Euclidean) distance

\[ d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2} \]
L1 (Manhattan) distance

\[ d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p| \]

K = 1

L2 (Euclidean) distance

\[ d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2} \]

K = 1
Hyperparameters

What is the best value of $k$ to use?
What is the best distance to use?

These are hyperparameters: choices about the algorithm that we set rather than learn
Hyperparameters

What is the best value of $k$ to use?
What is the best distance to use?

These are hyperparameters: choices about the algorithm that we set rather than learn

Very problem-dependent.
Must try them all out and see what works best.
Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

Your Dataset
Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: $K = 1$ always works perfectly on training data

Your Dataset
Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

| train   | test  |
Setting Hyperparameters

**Idea #1**: Choose hyperparameters that work best on the data

BAD: $K = 1$ always works perfectly on training data

---

**Idea #2**: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data
# Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data  
**BAD:** $K = 1$ always works perfectly on training data

![](Your Dataset)

**Idea #2:** Split data into `train` and `test`, choose hyperparameters that work best on test data  
**BAD:** No idea how algorithm will perform on new data

![](train) ![test]

**Idea #3:** Split data into `train`, `val`, and `test`; choose hyperparameters on `val` and evaluate on `test`  
**Better!**

![](train) ![validation] ![test]
Setting Hyperparameters

Idea #4: Cross-Validation: Split data into folds, try each fold as validation and average the results

- fold 1
- fold 2
- fold 3
- fold 4
- fold 5
- test

Useful for small datasets, but not used too frequently in deep learning
Setting Hyperparameters

Example of 5-fold cross-validation for the value of $k$.

Each point: single outcome.

The line goes through the mean, bars indicated standard deviation

(Seems that $k \sim 7$ works best for this data)
Nearest Neighbor - not used for images :-)

- Very slow at test time
- Distance metrics on pixels are not informative

(all 3 images have same L2 distance to the one on the left)
K-Nearest Neighbors: Summary

In **image classification** we start with a **training set** of images and labels, and must predict labels on the **test set**

The **K-Nearest Neighbors** classifier predicts labels based on nearest training examples

Distance metric and K are **hyperparameters**

Choose hyperparameters using the **validation set**; only run on the test set once at the very end!
Linear Classification

Neural Network

Linear classifiers
Recall CIFAR10

50,000 training images
each image is 32x32x3

10,000 test images.
Parametric Approach

Image

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

$f(x, W) \rightarrow$ 10 numbers giving class scores

$W$

parameters or weights
Parametric Approach: Linear Classifier

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

Image

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

\[ W \]

parameters or weights

10 numbers giving class scores
Parametric Approach: Linear Classifier

$f(x, W) = Wx$

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

$W$ parameters or weights

10 numbers giving class scores
Parametric Approach: Linear Classifier

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

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

$W$ parameters or weights

10 numbers giving class scores

Image
Parametric Approach: Linear Classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)
Parametric Approach: Linear Classifier

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

Input image:

\[
\begin{array}{c}
\text{56} \\
\text{231} \\
\text{24} \\
\text{2}
\end{array}
\]

Stretch pixels into column:

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

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

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

\[
= \begin{pmatrix}
56 \\
231 \\
24 \\
2
\end{pmatrix} + \begin{pmatrix}
1.1 \\
3.2 \\
-1.2
\end{pmatrix} = \begin{pmatrix}
61.95
\end{pmatrix}
\]

Cat score: -96.8
Dog score: 437.9
Ship score: 61.95
Parametric Approach: Linear Classifier

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

**Algebraic Viewpoint**

\[ f(x,W) = Wx \]
Parametric Approach: Linear Classifier

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

Algebraic Viewpoint

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

Input image

<table>
<thead>
<tr>
<th>W</th>
<th>b</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>-0.5</td>
<td>-96.8</td>
</tr>
<tr>
<td>0.1</td>
<td>2.0</td>
<td>437.9</td>
</tr>
<tr>
<td>1.5</td>
<td>1.3</td>
<td>61.95</td>
</tr>
<tr>
<td>0</td>
<td>0.25</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Cat score

Dog score

Ship score
Interpreting a Linear Classifier
Interpreting a Linear Classifier: Visual Viewpoint

airplane  automobile  bird  cat  deer  dog  frog  horse  ship  truck

Input image

W
0.2  0.5
0.1  2.0
1.5  1.3
2.1  0.0
0.2  -0.3

Score
-96.8
437.9
61.95

b
1.1
3.2
-1.2

plane  car  bird  cat  deer  dog  frog  horse  ship  truck
Interpreting a Linear Classifier: Geometric Viewpoint

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

Array of 32x32x3 numbers (3072 numbers total)
Hard Cases for a Linear Classifier

**Class 1:**
First and third quadrants

**Class 2:**
Second and fourth quadrants

**Class 1:**
1 \(\leq L_2\) norm \(\leq 2\)

**Class 2:**
Everything else

**Class 1:**
Three modes

**Class 2:**
Everything else
Linear Classifier: Three Viewpoints

**Algebraic Viewpoint**

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

**Visual Viewpoint**

One template per class

**Geometric Viewpoint**

Hyperplanes cutting up space
Linear Classifier…

So far: Defined a (linear) score function $f(x, W) = Wx + b$

Example class scores for 3 images for some $W$:

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

How can we tell whether this $W$ is good or bad?
Linear Classifier...

<table>
<thead>
<tr>
<th>Category</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td>-3.45</td>
</tr>
<tr>
<td>automobile</td>
<td>-8.87</td>
</tr>
<tr>
<td>bird</td>
<td>0.09</td>
</tr>
<tr>
<td>cat</td>
<td>2.9</td>
</tr>
<tr>
<td>deer</td>
<td>4.48</td>
</tr>
<tr>
<td>dog</td>
<td>8.02</td>
</tr>
<tr>
<td>frog</td>
<td>3.78</td>
</tr>
<tr>
<td>horse</td>
<td>1.06</td>
</tr>
<tr>
<td>ship</td>
<td>-0.36</td>
</tr>
<tr>
<td>truck</td>
<td>-0.72</td>
</tr>
</tbody>
</table>

5.11

52

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)
Example

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

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>

A loss function tells how good our current classifier is given a dataset of examples:

$$\left\{ \left( x_i, y_i \right) \right\}_{i=1}^{N}$$

Where $x_i$ is image and $y_i$ is (integer) label.

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_{i} L_i(f(x_i, W), y_i)$$
Multiclass Support Vector Machine (SVM) 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>

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:

\[
L_i = \sum_{j \neq y_i} \begin{cases} 
0 & \text{if } s_{y_i} \geq s_j + 1 \\
 s_j - s_{y_i} + 1 & \text{otherwise}
\end{cases} 
= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
\]
Multiclass Support Vector Machine (SVM) 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>1</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Multiclass SVM loss:

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

“Hinge loss”
Multiclass Support Vector Machine (SVM) 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>Losses:</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:

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

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

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

$$= 2.9 + 0$$

$$= 2.9$$
Multiclass Support Vector Machine (SVM) Loss

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</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:

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

$$= \max(0, 1.3 - 4.9 + 1)$$
$$+ \max(0, 2.0 - 4.9 + 1)$$
$$= \max(0, -2.6) + \max(0, -1.9)$$
$$= 0 + 0$$
$$= 0$$
Multiclass Support Vector Machine (SVM) Loss

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

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:

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

$$= \max(0, 2.2 - (-3.1) + 1)$$

$$+ \max(0, 2.5 - (-3.1) + 1)$$

$$= \max(0, 6.3) + \max(0, 6.6)$$

$$= 6.3 + 6.6$$

$$= 12.9$$
Multiclass Support Vector Machine (SVM) 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>Score</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>Loss</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>
<tr>
<td>Losses</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 \( y_i \) 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:

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

\[
L = (2.9 + 0 + 12.9)/3
\]

\[
= 5.27
\]
Multiclass Support Vector Machine (SVM) Loss

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</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:

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

Q: What happens to loss if car scores change a bit?
Multiclass Support Vector Machine (SVM) Loss

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

- **Cat**: 3.2 1.3 2.2
- **Car**: 5.1 4.9 2.5
- **Frog**: -1.7 2.0 -3.1

**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:

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

Q2: what is the min/max possible loss?
Multiclass Support Vector Machine (SVM) Loss

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

- **cat**
  - 3.2
  - 1.3
  - 2.2

- **car**
  - 5.1
  - 4.9
  - 2.5

- **frog**
  - -1.7
  - 2.0
  - -3.1

**Losses:**
- 2.9
- 0
- 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:

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

Q3: At initialization $W$ is small so all $s \approx 0$. What is the loss?
Multiclass Support Vector Machine (SVM) 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>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>

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:

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

Q4: What if the sum was over all classes? (including $j = y_i$)
### Multiclass Support Vector Machine (SVM) 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>Score</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>Loss</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>

**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:

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

**Q5:** What if we used mean instead of sum?
Multiclass Support Vector Machine (SVM) 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>
<tr>
<td>Losses:</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:

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

Q6: What if we used

\[
L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2
\]
\[ f(x, W) = Wx \]

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

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

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

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>$f(x, W)$</th>
<th></th>
<th>$f(x, W)$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>car</td>
<td>5.1</td>
<td>frog</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td></td>
<td>4.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td></td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>Losses:</td>
<td>2.9</td>
<td>Losses:</td>
<td>0</td>
<td>Losses:</td>
</tr>
</tbody>
</table>

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

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! \( 2W \) is also has \( L = 0 \! \! \) 
How do we choose between \( W \) and \( 2W \)?
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

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

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

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

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

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

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

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

- **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

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

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

L2 Regularization

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

L2 regularization likes to “spread out” the weights

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

\[ R(w_1) = 1^2 + 0^2 + \cdots = 1^2 \]
\[ R(w_2) = 0.25^2 + 0.25^2 + \cdots = 4 \times 0.25^2 = 0.25 \]
Softmax Classifier

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

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as *probabilities*

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

<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

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

Pre-normalized probabilities must be \( \geq 0 \)

\[
\begin{array}{c|c|c}
\text{cat} & 3.2 & 24.5 \\
\text{car} & 5.1 & 164.0 \\
\text{frog} & -1.7 & 0.18 \\
\end{array}
\]

unnormalized probabilities
Softmax Classifier (Multinomial Logistic Regression)

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

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

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

Softmax Function

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

Probabilities must be \( \geq 0 \)

Probabilities must sum to 1
**Softmax Classifier (Multinomial Logistic Regression)**

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

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

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

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

**Unnormalized log-probabilities / logits**

**Unnormalized probabilities**

**Probabilities**

Probabilities must be >= 0

Probabilities must sum to 1

Softmax Function
**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**

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>3.2</th>
<th>24.5</th>
<th>0.13</th>
</tr>
</thead>
<tbody>
<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>

Unnormalized log-probabilities / logits

Unnormalized probabilities

**Maximum Likelihood Estimation**

Choose probabilities to maximize the likelihood of the observed data

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

\[ L_i = -\log(0.13) = 2.04 \]
**Softmax Classifier (Multinomial Logistic Regression)**

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

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

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

Softmax Function

Probabilities must be \( \geq 0 \)

Probabilities must sum to 1

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

<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>

Unnormalized log-probabilities / logits

unnormalized probabilities

probabilities

Correct probs

**compare**
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

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

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

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

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

Kullback–Leibler divergence

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

Unnormalized log-probabilities / logits

Probabilities must be \( >= 0 \)

Probabilities must sum to 1

compare

Correct probs

cat

3.2

24.5

0.13

1.00

car

5.1

164.0

0.87

0.00

frog

-1.7

0.18

0.00

0.00

Unnormalized probabilities

probabilities

normalize

compare
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**

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

Cross Entropy

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

<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>

Unnormalized log-probabilities / logits → \(\exp\) → \text{unnormalized probabilities} → normalize → \text{probabilities} → compare → Correct probs
Let's double check...

- Entropy & KL-divergence:

\[
H(P^t) = - \sum_y P^t(y) \log P^t(y)
\]

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

- Cross Entropy the sum of both:

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

\[
= \sum_y P^t(y) \left( \log \frac{P^t(y)}{Q(y)} - \log P^t(y) \right)
\]

\[
= - \sum_y P^t(y) \log Q(y)
\]
Let's double check...

- Cross Entropy in our classification case:
  - Target "distribution" / output:
    \[ P^t(y) = \begin{cases} 
    1 & y = y_i \\
    0 & y \neq y_i 
    \end{cases} \]
  - Output of the network:
    \[ Q(y|x_i) = P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]
- Then Cross Entropy Loss for image \( x_i \)
  \[ L_i = L(x_i) = -\sum_y P^t(y) \log Q(y|x_i) \]
  \[ = -1 \cdot \log Q(y_i|x_i) \]
  \[ = - \log P(Y = y_i|X = x_i) \]
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities:

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

Softmax Function

Maximize probability of correct class:

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

Putting it all together:

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

- **cat**  \[ 3.2 \]
- **car**  \[ 5.1 \]
- **frog**  \[ -1.7 \]
Softmax Classifier (Multinomial Logistic Regression)

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

\[ s = f(x_i; W) \quad \text{Softmax Function} \]

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

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 \( L_i \)?

**Values:**
- cat: 3.2
- car: 5.1
- frog: -1.7
Softmax Classifier (Multinomial Logistic Regression)

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

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

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

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
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

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

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?

cat \hspace{0.5cm} 3.2

car \hspace{0.5cm} 5.1

frog \hspace{0.5cm} -1.7
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

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

Maximize probability of correct class

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

Putting it all together:

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

<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>

Q2: At initialization all \( s \) will be approximately equal; what is the loss?

A: \( \log(C) \), eg \( \log(10) \approx 2.3 \)
Softmax vs. SVM

matrix multiply + bias offset

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

hinge loss (SVM)

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

cross-entropy loss (Softmax)

\[
\begin{align*}
\exp(0.058) &= 2.36 \\
\exp(0.28) &= 1.32 \\
-\log(0.353) &= 0.452
\end{align*}
\]
Recap...

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

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

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

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)
\]

Softmax

SVM

Full loss
Recap

How do we find the best $W$?

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

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

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

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

**Softmax**  
**SVM**  
**Full loss**
Optimization
Strategies

• #1 Random Search
  ▸ can work…
  ▸ but very expensive

• #2 “Follow the Slope”
  ▸ aka: gradient descent…
Gradient Calculation (numeric)

In 1-dimensional, the derivative of a function:

$$\frac{df(W)}{dW} = \lim_{h \to \infty} \frac{f(W + h) - f(W)}{h}$$

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**.
Gradient Calculation (numeric)

<table>
<thead>
<tr>
<th>current W:</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>[?, ?, ?, ?, ?, ?, ?, ?, ?,...]</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td></td>
</tr>
</tbody>
</table>
## Gradient Calculation (numeric)

<table>
<thead>
<tr>
<th>current W:</th>
<th>( \mathbf{W} + \mathbf{h} ) (first dim):</th>
<th>gradient d( \mathbf{W} ):</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 + 0.0001, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33, ...]</td>
<td>[?, ?, ?, ?, ?, ?, ?, ?, ?,...]</td>
</tr>
<tr>
<td><strong>loss 1.25347</strong></td>
<td><strong>loss 1.25322</strong></td>
<td></td>
</tr>
</tbody>
</table>
Gradient Calculation (numeric)

<table>
<thead>
<tr>
<th>current $W$:</th>
<th>$W + h$ (first 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 + 0.0001, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>$[-2.5, ?, ?, (1.25322 - 1.25347)/0.0001 = -2.5]$</td>
</tr>
</tbody>
</table>

loss 1.25347  
loss 1.25322
Gradient Calculation (numeric)

<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, 0.6, ?, ?, (1.25353 - 1.25347)/0.0001 = 0.6, ?,...]</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25353</td>
<td></td>
</tr>
</tbody>
</table>

\[
\frac{df(W)}{dW} = \lim_{h \to \infty} \frac{f(W + h) - f(W)}{h}
\]
Gradient Calculation (numeric)

<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, ?, ?,...]</td>
</tr>
</tbody>
</table>

\[
\frac{df(W)}{dW} = \lim_{h \to \infty} \frac{f(W + h) - f(W)}{h} = 0
\]

\[(1.25347 - 1.25347)/0.0001 = 0\]
Gradient Calculation (numeric)

<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, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]</td>
</tr>
</tbody>
</table>

**Numeric Gradient**
- Slow! Need to loop over all dimensions
- Approximate
Gradient Calculation

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
Gradient Calculation

current $W$: 

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

loss $1.25347$

$\text{gradient } dW$: 

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

d$W = \ldots$

(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

**General gradient descent:** Start with initial point $x_0$,

\[ W_{t+1} = W_t + \alpha_t d_t \]

**Steepest Descent:**

\[ d_t = -\nabla f(W_t) \text{ (we move in the opposite direction of the gradient).} \]

---

# Vanilla Gradient Descent

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

---

![Diagram of gradient descent](image_url)
Gradient Descent

W_2

W_1

original W

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

```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
```
Aside: Image Features

\[ f(x) = Wx \]

Class scores
Aside: Image Features

\[ f(x) = Wx \]

Feature Representation

Class scores
Image Features: Motivation

Cannot separate red and blue points with linear classifier

x

y
Image Features: Motivation

\[ f(x, y) = (r(x, y), \theta(x, y)) \]

Cannot separate red and blue points with linear classifier

After applying feature transform, points can be separated by linear classifier
Example: Color Histogram
Example; Histogram of Oriented Gradients (HOG)

Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins

Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has 30*40*9 = 10,800 numbers

Lowe, "Object recognition from local scale-invariant features", ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005
Example: Bag of Words:

**Step 1: Build codebook**

- Extract random patches
- Cluster patches to form “codebook” of “visual words”

**Step 2: Encode images**

Fei-Fei and Perona, "A bayesian hierarchical model for learning natural scene categories", CVPR 2005
Aside: Image Features
Image Features vs. ConvNets

Feature Extraction

10 numbers giving scores for classes

training