Image Classification

Input Image

Output Class

- dog
- cat
- mountain
- ship

- dog
- cat
- mountain
- ship
Example dataset: CIFAR10

Images: 32x32x3 (RGB)

10 classes

Training set: 5k images per class

Test set: 1k images per class

Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009
K-Nearest Neighbor Classification
How do we calculate the distance between two images?

Square of difference

\[
\begin{array}{cc}
225 & 2809 \\
49 & 1849 \\
\end{array}
\]

Add and take square root

\[
= \sqrt{225 + 2809 + 49 + 1849} \\
= 70.23
\]

Called the L2 Distance between two images.
L1 distance between two images

Called the L1 Distance between two images.

Absolute differences

\[
\begin{array}{cc}
15 & 53 \\
7 & 43 \\
\end{array}
\]

Add the absolute differences

\[
= 15 + 53 + 7 + 43 \\
= 118
\]
Recap: How do we choose k?

• The goal of a machine learning system is for the system to generalize in the wild by training on a training dataset which is hopefully representative of the samples to which we want to apply our system.

• How do we get a measure of how good our system will be?

• We calculate its accuracy on a “test set” which is done only once at the end of training.

• How do we tune our hyperparameters (like ‘k’) then?
Training, Validation and Test Set

Choose **hyperparameters** like k, feature space, similarity function, etc.

Pool of nearest neighbors

% accuracy

choose best

\( k = 3 \)

Training Set

Validation Set

Test Set

Measures generalization. Ideally only test once at end!
K-Nearest Neighbors: Web Demo

Interactively move points around and see decision boundaries change

Play with L1 vs L2 metrics

Play with changing number of training points, value of K

http://vision.stanford.edu/teaching/cs231n-demos/knn/

Slide credits: Justin Johnson
Linear classifiers
Linear Classifier

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

\[ x : (32x32x3) \]

\[ W: (10, 3072) \]
\[ b: (10, ) \]

250.75
30.5
67.325
-20.25

\[ \text{dog} \]
\[ \text{cat} \]
\[ \text{mountain} \]
\[ \text{ship} \]
Example 2x2 image, 3 classes (dog, cat, mountain)

Idea credit: Justin Johnson
### Bias Trick

**Flatten the image**

<table>
<thead>
<tr>
<th>25</th>
<th>243</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W</th>
<th>(3, 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.2</td>
</tr>
<tr>
<td>2.6</td>
<td>-1.2</td>
</tr>
<tr>
<td>0.25</td>
<td>0.0</td>
</tr>
<tr>
<td>0.7</td>
<td>-2.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>(5, )</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>243</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Scores (s)**

<table>
<thead>
<tr>
<th>(3, )</th>
</tr>
</thead>
<tbody>
<tr>
<td>73.1</td>
</tr>
<tr>
<td>-201.6</td>
</tr>
<tr>
<td>51.65</td>
</tr>
</tbody>
</table>

Extra ‘1’ appended to account for bias
Visualizing the weights

Idea credits: Justin Johnson
Visualizing weights for CIFAR10

The ‘unflattened’ weights can be visualized as an image

Linear classifiers have a “single” template per category

You will get to see similar visualizations after training your own linear classifier in PS3
Mean images from training set of CIFAR10

Learned weights for CIFAR10
Softmax loss / Multinomial logistic loss

For a single image, \( x \) with label ‘y’

\[
L(s, y) = -\log \frac{e^{sy}}{\sum_{j=1}^{C} e^{sj}}
\]

Scores / Logits, \( s = Wx \)

Note: In class slides, logits are denoted by ‘z’.

For class y = 1 (Dog):

\[
L(s, 1) = -\log 0.06 = 2.81
\]

Probabilities

\[
\frac{e^{sk}}{\sum_{j=1}^{3} e^{sj}}
\]

\[
\begin{array}{c|c|c}
\text{Scores} & 2.5 & 12.18 \\
\text{Logits} & -1.3 & 0.27 \\
\text{Exponent} & 5.2 & 181.27 \\
\text{Normalized} & 0.06 & 0.0 \\
\end{array}
\]
Softmax loss properties

\[ L(s, y) = - \log \frac{e^{sy}}{\sum_{j=1}^{C} e^{sj}} \]

C: Number of classes

What is the min/max possible Value of \( L(s, y) \)?

Min 0, Max +infinity

If all scores were really small and thus approximately the same, What would be the value of \( L(s, y) \)?

\( \log(C) \)
How do we learn optimal parameters?

Review from Class

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} L(f_\theta(x_i), y_i) \]

\[ J(\theta) \]

\[ \theta^{t+1} = \theta^t - \eta_t \nabla_{\theta} J(\theta) \bigg|_{\theta = \theta^t} \]
Recap: Stochastic Gradient Descent

\[ \nabla J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(x_i, y_i, \theta) \]

Vanilla Gradient Descent

We average the gradients over the whole training set which can be expensive for a large training set. (Large N).

- Stochastic Gradient descent: Approximate this sum using a minibatch of training examples, which are a random subset of a fixed size \(B\).

\[ \nabla J(\theta) \approx \frac{1}{|B|} \sum_{i \in B} \nabla L(x_i, y_i, \theta) \]

\(B\) – Batch Size
Calculating gradients using finite differences

\[
\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x + \epsilon) - f(x)}{\epsilon}
\]

• Called **numeric gradient**.
current $W$:  

\[
\begin{bmatrix}
0.34, \\
-1.11, \\
0.78, \\
0.12, \\
0.55, \\
2.81, \\
-3.1, \\
-1.5, \\
0.33, \ldots
\end{bmatrix}
\]

loss 1.25347

$W + h$ (first dim):  

\[
\begin{bmatrix}
0.34 + 0.0001, \\
-1.11, \\
0.78, \\
0.12, \\
0.55, \\
2.81, \\
-3.1, \\
-1.5, \\
0.33, \ldots
\end{bmatrix}
\]

loss 1.25322

gradient $dL/dW$:  

\[
\begin{bmatrix}
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \ldots
\end{bmatrix}
\]
current W:

\[
\begin{bmatrix}
0.34, \\
-1.11, \\
0.78, \\
0.12, \\
0.55, \\
2.81, \\
-3.1, \\
-1.5, \\
0.33, ...
\end{bmatrix}
\]

loss 1.25347

\[W + h \text{ (first dim):}\]

\[
\begin{bmatrix}
0.34 + 0.0001, \\
-1.11, \\
0.78, \\
0.12, \\
0.55, \\
2.81, \\
-3.1, \\
-1.5, \\
0.33, ...
\end{bmatrix}
\]

loss 1.25322

\[\text{gradient } dL/dW:\]

\[
\begin{bmatrix}
-2.5, \\
?, \\
?, \\
(1.25322 - 1.25347)/0.0001 = -2.5 \\
?, \\
?, ...
\end{bmatrix}
\]

\[
\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x + \epsilon) - f(x)}{\epsilon}
\]
How to analytically calculate gradients?

- Numeric gradients are easy to compute.

- But they are slow and approximate.

- Useful for gradient checking. We have implemented a function which does this and checks if the gradients implemented by you are correct.

- How do we efficiently calculate gradients analytically? We do it using a method called Backpropagation. Stay tuned to learn about this next week in class!