Convolutional neural networks II

September 30th, 2019

Yong Jae Lee
UC Davis

Many slides from Rob Fergus, Svetlana Lazebnik, Jia-Bin Huang, Derek Hoiem, Adriana Kovashka, Andrej Karpathy
Announcements

• Sign-up for paper presentations
Standard classifiers

Nearest neighbor

10^6 examples

Shakhnarovich, Viola, Darrell 2003
Berg, Berg, Malik 2005...

Neural networks

LeCun, Bottou, Bengio, Haffner 1998
Rowley, Baluja, Kanade 1998
...

Support Vector Machines

Guyon, Vapnik
Heisele, Serre, Poggio, 2001,...

Boosting

Viola, Jones 2001,
Torralba et al. 2004,
Opelt et al. 2006,...

Conditional Random Fields

McCallum, Freitag, Pereira 2000; Kumar, Hebert 2003
...

Slide adapted from Antonio Torralba
Deep neural networks

- Lots of hidden layers
- Depth = power (usually)

Figure from http://neuralnetworksanddeeplearning.com/chap5.html
How do we train them?

• The goal is to iteratively find a set of weights that allow the activations/outputs to match the desired output

• For this, we will minimize a loss function

• The loss function quantifies the agreement between the predicted scores and GT labels

• First, let’s simplify and assume we have a single layer of weights in the network
Classification goal

Example dataset: CIFAR-10
10 labels
50,000 training images
each image is 32x32x3
10,000 test images.
Classification scores

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

\[ f(x, W) \]

[32x32x3] array of numbers 0...1 (3072 numbers total)

10 numbers, indicating class scores
Linear classifier

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

- \( f(x, W) \) is a function that takes an input \( x \) and parameters \( W \) and returns an output.
- \( W \) is a matrix of parameters or "weights".
- \( x \) is an array of numbers representing an input.
- \( b \) is a vector of 10 numbers, indicating class scores.

[32x32x3] array of numbers 0...1

10 numbers, indicating class scores

parameters, or "weights"
Linear classifier

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

<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>scores</td>
<td>-3.45</td>
<td>-8.87</td>
<td>0.09</td>
<td>2.9</td>
<td>4.48</td>
<td>8.02</td>
<td>3.78</td>
<td>1.06</td>
<td>-0.36</td>
<td>-0.72</td>
</tr>
<tr>
<td></td>
<td>-0.51</td>
<td>6.04</td>
<td>5.31</td>
<td>-4.22</td>
<td>-4.19</td>
<td>3.58</td>
<td>4.49</td>
<td>-4.37</td>
<td>-2.09</td>
<td>-2.93</td>
</tr>
<tr>
<td></td>
<td>3.42</td>
<td>4.64</td>
<td>2.65</td>
<td>5.1</td>
<td>2.64</td>
<td>5.55</td>
<td>-4.34</td>
<td>-1.5</td>
<td>-4.79</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)*
Linear classifier

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

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

Adapted from Andrej Karpathy
Linear classifier: 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>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>

Hinge loss:

Given an example $\left(x_i, y_i\right)$ 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 loss has the form:

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

Want: $s_{y_i} \geq s_j + 1$

i.e. $s_j - s_{y_i} + 1 \leq 0$

If true, loss is 0
If false, loss is magnitude of violation

Adapted from Andrej Karpathy
Linear classifier: 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></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>

Loss: **2.9**

Hinge 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 loss has the form:

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

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

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

$$= 2.9 + 0$$

$$= 2.9$$

Adapted from Andrej Karpathy
Linear classifier: 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></td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</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>

Hinge 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 loss has the form:

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

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

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

$$= 0 + 0$$

$$= 0$$

Adapted from Andrej Karpathy
Linear classifier: 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>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>

Loss:

- cat: $2.9$
- car: $0$
- frog: $12.9$

Hinge 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 loss has the form:

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

$$= \max(0, 2.2 - (-3.1) + 1)$$
$$+ \max(0, 2.5 - (-3.1) + 1)$$
$$= \max(0, 5.3 + 1)$$
$$+ \max(0, 5.6 + 1)$$
$$= 6.3 + 6.6$$
$$= 12.9$$

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

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

**Loss:**

$2.9$ $0$ $12.9$

**Hinge 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 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:

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

$L = (2.9 + 0 + 12.9)/3 = 15.8 / 3 = 5.3$
Linear classifier: Hinge loss

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

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

Adapted from Andrej Karpathy
Linear classifier: Hinge loss

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
- L1 regularization
- Dropout

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

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

Adapted from Andrej Karpathy
Another loss: Softmax (cross-entropy)

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:

\[ L_i = -\log P(Y = y_i|X = x_i) \]
Another loss: Softmax (cross-entropy)

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

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

\[ L_i = -\log(0.13) = 0.89 \]

Adapted from Andrej Karpathy
How to minimize the loss function?
How to minimize the loss function?

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:**

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

**loss** 1.25347

**gradient dW:**

\[
\begin{bmatrix}
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?,
\end{bmatrix}
\]
<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>$[0.34 + \text{0.0001},$</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>
</tbody>
</table>
current W: [0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33, ...]  
W + h (first dim): [0.34 + 0.0001, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33, ...]  
loss 1.25347  
loss 1.25322  

gradient dW:  
[-2.5, ?, ?, ?, ?, 1.25322 - 1.25347)/0.0001 = -2.5  
 Andrej Karpathy
<table>
<thead>
<tr>
<th>current W:</th>
<th>( W + h ) (second dim):</th>
<th>gradient dW:</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>
| loss 1.25347 | loss 1.25353 | \[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]
| | | \[
\frac{(1.25353 - 1.25347)}{0.0001} = 0.6
\]
<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>
</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!

$$\nabla_W L = ...$$
current $W$: 

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

\textbf{loss} 1.25347

gradient $dW$: 

\[ \begin{array}{c}
-2.5, \\
0.6, \\
0, \\
0.2, \\
0.7, \\
-0.5, \\
1.1, \\
1.3, \\
-2.1, \\
\end{array} \ldots \]

dW = ... 
(some function data and W)
Loss gradients

• Denoted as (diff notations): \( \frac{\partial E}{\partial w^{(1)}_{ji}} \) $\nabla_W L$

• i.e. how the loss changes as a function of the weights

• We want to change the weights in such a way that makes the loss decrease as fast as possible
Gradient descent

- We’ll update weights iteratively
- Move in direction opposite to gradient:

\[ w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)}) \]

Figure from Andrej Karpathy
Gradient descent

- Iteratively *subtract* the gradient with respect to the model parameters (w)
- i.e. we’re moving in a direction opposite to the gradient of the loss
- i.e. we’re moving towards *smaller* loss
Mini-batch gradient descent

• In classic gradient descent, we compute the gradient from the loss for all training examples (can be slow)
• So, use only use *some* of the data for each gradient update
• We cycle through all the training examples multiple times
• Each time we’ve cycled through all of them once is called an ‘epoch’
Learning rate selection

The effects of step size (or “learning rate”)

- Good learning rate
- Low learning rate
- High learning rate
- Very high learning rate

Loss vs. Epoch
Questions?

See you Wednesday!