Lecture 23: Optimization and Neural Nets
Today

• Optimization

• Today and Thursday: Neural nets, CNNs
  – Mon: http://cs231n.github.io/classification/
  – Today:
    • http://cs231n.github.io/optimization-1/
    • http://cs231n.github.io/optimization-2/
Announcements

• Final project (P5) released, due Tuesday, 5/9, by 11:59pm, to be done in groups of two

• Final exam will be handed out in class Tuesday, due Friday, 5/12, by 5pm

• Project 3 voting results
Third Place
Second Place
Arpit Sabherwal and Jaldeep Acharya
First Place
Hong Gan and Renkai Xiang
Summary

1. Score function

\[ f(x_i, W, b) = Wx_i + b \]

2. Loss function

\[ L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[ \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda R(W) \]
Other loss functions

• Scores are not very intuitive

• Softmax classifier
  – Score function is same
  – Intuitive output: normalized class probabilities
  – Extension of logistic regression to multiple classes
Softmax classifier

\[ f(x_i, W) = Wx_i \]

score function is the same

\[
\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}
\]

softmax function

\[
[1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26]
\]

Interpretation: squashes values into range 0 to 1

\[ P(y_i | x_i; W) \]
Cross-entropy loss

\[ f(x_i, W) = W x_i \]

score function is the same

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

\[ L_i = -f_{y_i} + \log \sum_j e^{f_j} \]

i.e. we’re minimizing the negative log likelihood.
Aside: Loss function interpretation

• Probability
  – Maximum Likelihood Estimation (MLE)
  – Regularization is Maximum a posteriori (MAP) estimation

• Cross-entropy \( H \)
  – \( p \) is true distribution (1 for the correct class), \( q \) is estimated
  – Softmax classifier minimizes cross-entropy
  – Minimizes the KL divergence (Kullback-Leibler) between the distribution: distance between \( p \) and \( q \)

\[
H(p, q) = - \sum_x p(x) \log q(x)
\]
Example of the difference between the SVM and Softmax classifiers for one datapoint. In both cases we compute the same score vector $f$ (e.g. by matrix multiplication in this section). The difference is in the interpretation of the scores in $f$: The SVM interprets these as class scores and its loss function encourages the correct class (class 2, in blue) to have a score higher by a margin than the other class scores. The Softmax classifier instead interprets the scores as (unnormalized) log probabilities for each class and then encourages the (normalized) log probability of the correct class to be high (equivalently the negative of it to be low). The final loss for this example is 1.58 for the SVM and 1.04 for the Softmax classifier, but note that these numbers are not comparable; They are only meaningful in relation to loss computed within the same classifier and with the same data.
Summary

• Have score function and loss function
  – Will generalize the score function

• Find W and b to minimize loss
  – SVM vs. Softmax
    • Comparable in performance
    • SVM satisfies margins, softmax optimizes probabilities

\[
L = \frac{1}{N} \sum \sum \left[ \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda \sum \sum W_{k,l}^2
\]

\[
L = \frac{1}{N} \sum \log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) + \lambda \sum \sum W_{k,l}^2
\]
Step size: learning rate
Too big: will miss the minimum
Too small: slow convergence
Analytic Gradient

\[ L_i = \sum_{j \neq y_i} \left[ \max(0, w_j^T x_i - w_{y_i}^T x_i + 1) \right] \]

\[ \nabla_{w_j} L_i = 1(\omega_j^T x_i - \omega_{y_i}^T x_i + \Delta > 0) x_i \]

\[ \nabla_{w_{y_i}} L_i = -\left( \sum_{j \neq y_i} 1(\omega_j^T x_i - \omega_{y_i}^T x_i + \Delta > 0) \right) x_i \]

Full gradient is the sum of all \( L_i \)s over all training examples \( x_i \)
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

# Vanilla Gradient Descent

```python
while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad  # perform parameter update
```
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 ~100 examples.
e.g. Krizhevsky ILSVRC ConvNet used 256 examples
Stochastic Gradient Descent (SGD)

- use a single example at a time

```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
```

(also sometimes called on-line Gradient Descent)
Summary

- Always use mini-batch gradient descent
- Incorrectly refer to it as “doing SGD” as everyone else (or call it batch gradient descent)
- The mini-batch size is a hyperparameter, but it is not very common to cross-validate over it (usually based on practical concerns, e.g. space/time efficiency)
The dynamics of Gradient Descent

$$L = \frac{1}{N} \sum \sum_{i \neq y_i} \left[ \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta) \right] + \lambda \sum \sum_{k,l} W_{k,l}^2$$

pull some weights up and some down

$$L = \frac{1}{N} \sum_i - \log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) + \lambda \sum \sum_{k,l} W_{k,l}^2$$

always pull the weights down
Momentum Update

\[
\text{weights\_grad} = \text{evaluate\_gradient}(\text{loss\_fun, data, weights})
\]
\[
\text{vel} = \text{vel} \times 0.9 - \text{step\_size} \times \text{weights\_grad}
\]
\[
\text{weights} += \text{vel}
\]
Many other ways to perform optimization...

- Second order methods that use the Hessian (or its approximation): BFGS, LBFGS, etc.

- Currently, the lesson from the trenches is that well-tuned SGD+Momentum is very hard to beat for CNNs.
Where are we?

• Classifiers: SVM vs. Softmax

• Gradient descent to optimize loss functions
  – Batch gradient descent, stochastic gradient descent
  – Momentum
  – Numerical gradients (slow, approximate), analytic gradients (fast, error-prone)
Derivatives

• Given $f(x)$, where $x$ is vector of inputs
  – Compute gradient of $f$ at $x$: $\nabla f(x)$
Examples

\[ f(x, y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x \]

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

\[ f(x + h) = f(x) + h \frac{df(x)}{dx} \]
\[ f(x, y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x \]

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

\[ f(x + h) = f(x) + h \frac{df(x)}{dx} \]

Example: \( x = 4, \ y = -3 \). \[ \Rightarrow f(x, y) = -12 \]

\[ \frac{\partial f}{\partial x} = -3 \quad \frac{\partial f}{\partial y} = 4 \]

\[ \nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \]

partial derivatives

gradient
Compound expressions:

\[ f(x, y, z) = (x + y)z \]

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} \]
Now onto Deep Learning

When a user takes a photo, the app should check whether they're in a national park...

Sure, easy GIS lookup. Gimme a few hours.

...and check whether the photo is of a bird.

I'll need a research team and five years.

In CS, it can be hard to explain the difference between the easy and the virtually impossible.

[Monroe 2014, xkcd]
Introducing: Flickr PARK or BIRD

Zion National Park Utah by Les Haines (cc) BY

Secretary Bird by Bill Gracey (cc) BY-NC-ND

Slide: Flickr
To play, drag an image from the examples or from your desktop.

---

**PARK or BIRD**

Want to know if your photo is from a U.S. national park? Want to know if it contains a bird? Just drag it into the box to the left, and we’ll tell you. We will use the GPS embedded in your photo (if it’s there) to see whether it’s from a park, and we will use our super-cool computer vision skills to try to see whether it’s a bird (which is a hard problem, but we do a pretty good job at it).

To try it out, just drag any photo from your desktop into the upload box, or try dragging any of our example images. We’ll give you your answers below!

Want to know more about PARK or BIRD, including why the heck we did this? Just click here for more info ➔

---

**EXAMPLE PHOTOS**

---

**PARK?  BIRD?**

---

Photo credits
PARK or BIRD

Want to know if your photo is from a U.S. national park? Want to know if it contains a bird? Just drag it into the box to the left, and we'll tell you. We'll use the GPS embedded in your photo (if it's there) to see whether it's from a park, and we'll use our super-cool computer vision skills to try to see whether it's a bird (which is a hard problem, but we do a pretty good job at it).

To try it out, just drag any photo from your desktop into the upload box, or try dragging any of our example images. We'll give you your answers below!

Want to know more about PARK or BIRD, including why the heck we did this? Just click here for more info →

PARK? YES
Ah yes, Bryce Canyon is truly beautiful.

BIRD? NO
Beautiful clouds, but I don't see any birds flying up there.

Photo credits
PARK or BIRD

Want to know if your photo is from a U.S. national park? Want to know if it contains a bird? Just drag it into the box to the left, and we'll tell you. We'll use the GPS embedded in your photo (if it's there) to see whether it's from a park, and we'll use our super-cool computer vision skills to try to see whether it's a bird (which is a hard problem, but we do a pretty good job at it).

To try it out, just drag any photo from your desktop into the upload box, or try dragging any of our example images. We'll give you your answers below!

Want to know more about PARK or BIRD, including why the heck we did this? Just click here for more info →

PARK?

YES

Hey, yeah! I went to Everglades once!

BIRD?

YES

Hey! Nice bird shot!

Photo credits

Slide: Flickr
In the next week, we’ll learn what this is, how to compute it, and how to learn it.
What is a Convolutional Neural Network (CNN)?

Key questions:

- What kinds of functions should we use?
- How do we learn the parameters for those functions?
Example CNN

Conv \rightarrow ReLU \rightarrow Conv \rightarrow ReLU \rightarrow Pool \rightarrow Softmax \rightarrow Fully Connected

*This network is running live in your browser

[Andrej Karpathy]
CNNs in 1989: “LeNet”

CNNs were not invented overnight

LeNet: a classifier for handwritten digits. [LeCun 1989]
CNNs in 2012: “SuperVision” (aka “AlexNet”)

“AlexNet” — Won the ILSVRC2012 Challenge

Major breakthrough: 15.3% Top-5 error on ILSVRC2012 (Next best: 25.7%)

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

[Krizhevsky, Sutskever, Hinton. NIPS 2012]
CNNs in 2014: “GoogLeNet”

“GoogLeNet” — Won the ILSVRC2014 Challenge

6.67% top-5 error rate!
(1000 classes!)

[ Szegedy et al, arXiv 2014 ]
CNNs in 2014: “VGGNet”

“VGGNet” — Second Place in the ILSVRC2014 Challenge

<table>
<thead>
<tr>
<th>ConvNet Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
</tr>
<tr>
<td>11 weight layers</td>
</tr>
</tbody>
</table>

input (224 × 224 RGB image)

- conv3-64
- conv3-64
- conv3-64
- conv3-64
- conv3-64
- conv3-64

- LRN

- conv3-128
- conv3-128
- conv3-128
- conv3-128
- conv3-128
- conv3-128

- maxpool

- conv3-256
- conv3-256
- conv3-256
- conv3-256
- conv3-256
- conv3-256

- conv1-256

- maxpool

- conv3-512
- conv3-512
- conv3-512
- conv3-512
- conv3-512
- conv3-512

- conv1-512

- maxpool

- FC-4096
- FC-4096
- FC-1000

- soft-max

No fancy picture, sorry

7.3% top-5 error rate

(and 1st place in the detection challenge)

[Simonyan et al, arXiv 2014]
CNNs in 2015: “ResNet”

Note: Despite its massive depth, ResNet has a lower runtime complexity than VGG

https://youtu.be/1PGLj-uKT1w?t=4m40s

CNNs in 2015: “ResNet”

• **1st places in all five main tracks**
  • ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer** nets
  • ImageNet Detection: **16%** better than 2nd
  • ImageNet Localization: **27%** better than 2nd
  • COCO Detection: **11%** better than 2nd
  • COCO Segmentation: **12%** better than 2nd

Aside: Before Deep Learning

Input Pixels → Extract Features → Concatenate into a vector $\mathbf{x}$ → Linear Classifier → SVM → Ans

Figure: Karpathy 2016
Why use features? Why not pixels?

Q: What would be a very hard set of classes for a linear classifier to distinguish?

(assuming $x = \text{pixels}$)
The last layer of (most) CNNs are linear classifiers

This piece is just a linear classifier

Input Pixels  Perform everything with a big neural network, trained end-to-end

Key: perform enough processing so that by the time you get to the end of the network, the classes are linearly separable
Linearly separable classes

\[ f(x_i, W, b) = Wx_i + b \]
Example: Visualizing AlexNet in 2D with t-SNE

(2D visualization using t-SNE)

Questions?