CS 2770: Computer Vision

Convolutional Neural Networks

Prof. Adriana Kovashka
University of Pittsburgh
February 20, 2018
Plan for the next few lectures

Why (convolutional) neural networks?

Neural network basics
• Architecture
• Biological inspiration
• Loss functions
• Optimization / gradient descent
• Training with backpropagation

Convolutional neural networks (CNNs)
• Special operations
• Common architectures

Practical matters
• Tips and tricks for training
• Transfer learning
• Software packages

Understanding CNNs
• Visualization
• Synthesis / style transfer
• Breaking CNNs
Neural network basics
Why (convolutional) neural networks?

State of the art performance on many problems
Most papers in recent vision conferences use deep neural networks

Razavian et al., CVPR 2014 Workshops
ImageNet Challenge 2012

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk
- Challenge: 1.2 million training images, 1000 classes

[Deng et al. CVPR 2009]

AlexNet: Similar framework to LeCun’98 but:
- Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
- More data (10^6 vs. 10^3 images)
- GPU implementation (50x speedup over CPU)
  - Trained on two GPUs for a week
- Better regularization for training (DropOut)


Adapted from Lana Lazebnik
ImageNet Challenge 2012

Krizhevsky et al. -- **16.4% error** (top-5)
Next best (non-convnet) – **26.2% error**
Example: CNN features for detection

R-CNN: Regions with CNN features

1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

Object detection system overview. Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs. **R-CNN achieves a mean average precision (mAP) of 53.7% on PASCAL VOC 2010.** For comparison, Uijlings et al. (2013) report 35.1% mAP using the same region proposals, but with a spatial pyramid and bag-of-visual-words approach. The popular deformable part models perform at 33.4%.


Lana Lazebnik
What are CNNs?

- Convolutional neural networks are a type of neural network with layers that perform special operations.
- Used in vision but also in NLP, biomedical etc.
- Often they are deep.
Traditional Recognition Approach

- Features are key to recent progress in recognition, but research shows they’re flawed…
- Where next? Better classifiers? Or keep building more features?

Adapted from Lana Lazebnik
What about learning the features?

- Learn a feature hierarchy all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly
“Shallow” vs. “deep” architectures

Traditional recognition: “Shallow” architecture

Image/Video Pixels → Hand-designed feature extraction → Trainable classifier → Object Class

Deep learning: “Deep” architecture

Image/Video Pixels → Layer 1 → ... → Layer N → Simple classifier → Object Class
Neural network definition

Figure 5.1 Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables $x_0$ and $z_0$. Arrows denote the direction of information flow through the network during forward propagation.

- **Activations:**
  \[ a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \]

- **Nonlinear activation function** $h$ (e.g. sigmoid, RELU):
  \[ z_j = h(a_j) \]

Recall SVM: $w^T x + b$
Neural network definition

• Layer 2

\[ a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \]

• Layer 3 (final)

• Outputs (e.g. sigmoid/softmax) (multiclass)

(binary) \[ y_k = \sigma(a_k) = \frac{1}{1 + \exp(-a_k)} \] (binary) \[ y_k = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \]

• Finally:

(binary) \[ y_k(x, w) = \sigma \left( \sum_{j=1}^{M} w_{kj}^{(2)} h_0 \left( \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \]
Activation functions

**Sigmoid**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

**tanh**

$$\tanh(x)$$

**ReLU**

$$\max(0, x)$$

**Leaky ReLU**

$$\max(0.1x, x)$$

**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

**ELU**

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$
A multi-layer neural network

- Nonlinear classifier
- Can approximate any continuous function to arbitrary accuracy given sufficiently many hidden units
Inspiration: Neuron cells

- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node
- If output of function over threshold, neuron “fires”
Biological analog

A biological neuron

An artificial neuron

Input

Weights

Output: $\sigma(w \cdot x + b)$

Sigmoid function:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Jia-bin Huang
Biological analog

Hubel and Weisel’s architecture

Multi-layer neural network

Adapted from Jia-bin Huang
Multilayer networks

- Cascade neurons together
- Output from one layer is the input to the next
- Each layer has its own sets of weights
Feed-forward networks

- Predictions are fed forward through the network to classify
Feed-forward networks

- Predictions are fed forward through the network to classify
Feed-forward networks

- Predictions are fed forward through the network to classify
Feed-forward networks

• Predictions are fed forward through the network to classify
Feed-forward networks

- Predictions are fed forward through the network to classify
Feed-forward networks

- Predictions are fed forward through the network to classify
Deep neural networks

- Lots of hidden layers
- Depth = power (usually)
How do we train them?

• The goal is to iteratively find such a set of weights that allow the activations/outputs to match the desired output
• We want to minimize a **loss function**
• The loss function is a function of the weights in the network
• For now let’s simplify and assume there’s 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

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

10 numbers, indicating class scores

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

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

A \([32x32x3]\) array of numbers 0...1

10x1 parameters, or "weights"

10 numbers, indicating class scores
Linear classifier

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

Going forward: Loss function/Optimization

<table>
<thead>
<tr>
<th></th>
<th>Score 1</th>
<th>Score 2</th>
<th>Score 3</th>
<th>Score 4</th>
<th>Score 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td>-3.45</td>
<td>-0.51</td>
<td>3.42</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>
</tr>
<tr>
<td>bird</td>
<td>0.09</td>
<td>5.31</td>
<td>2.65</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>
</tr>
<tr>
<td>deer</td>
<td>4.48</td>
<td>-4.19</td>
<td>2.64</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>
</tr>
<tr>
<td>frog</td>
<td>3.78</td>
<td>4.49</td>
<td>-4.34</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>
</tr>
<tr>
<td>ship</td>
<td>-0.36</td>
<td>-2.09</td>
<td>-4.79</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>
</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:

<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></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>
**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>

**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:

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

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></th>
<th>car</th>
<th></th>
<th>frog</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
<td></td>
<td>5.1</td>
<td>4.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Losses:</td>
<td>2.9</td>
<td></td>
<td></td>
<td></td>
<td></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:

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

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

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

$$= 2.9 + 0$$

$$= 2.9$$

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>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>

**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>3.2</td>
<td>1.3</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
<td></td>
</tr>
</tbody>
</table>

Losses: 2.9 0 12.9

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:

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

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

Adapted from Andrej Karpathy
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>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>
<tr>
<td>Losses</td>
<td>2.9</td>
<td>0</td>
<td>12.9</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:

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

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

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

$L = (2.9 + 0 + 12.9)/3$

$= 15.8 / 3 = 5.3$

Adapted from Andrej Karpathy
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

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

In common use:

**L2 regularization**

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

**L1 regularization**

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

 dropout (will see later)

λ = regularization strength (hyperparameter)

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:

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

<table>
<thead>
<tr>
<th></th>
<th>Scores</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>
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>
<tr>
<td>unnormalized log probabilities</td>
<td>24.5</td>
<td>164.0</td>
<td>0.18</td>
</tr>
<tr>
<td>normalize</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>probabilities</td>
<td>0.13</td>
<td>0.87</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Another loss:
Softmax (cross-entropy)

Adapted from Andrej Karpathy
Other losses

- **Triplet loss (Schroff, FaceNet)**

\[
\sum_{i=1}^{N} \left[ \| f(x^a_i) - f(x^p_i) \|^2_2 - \| f(x^a_i) - f(x^n_i) \|^2_2 + \alpha \right]_+
\]

a denotes anchor
p denotes positive
n denotes negative

Figure 3. The **Triplet Loss** minimizes the distance between an anchor and a positive, both of which have the same identity, and maximizes the distance between the anchor and a negative of a different identity.

- **Anything you want!**
How to minimize the loss function?
How to minimize the loss function?

In 1-dimensional, 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:

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

loss 1.25347

gradient dW:

\[
\begin{bmatrix}
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \\
?, \ldots
\end{bmatrix}
\]
<table>
<thead>
<tr>
<th>current $W$:</th>
<th>$W + h$ (first dim):</th>
<th>gradient $dW$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -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>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>[-2.5, ?, ?, (1.25322 - 1.25347)/0.0001 = -2.5]</td>
</tr>
<tr>
<td>loss 1.25347</td>
<td>loss 1.25322</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>loss 1.25347</td>
<td>loss 1.25353</td>
<td></td>
</tr>
</tbody>
</table>
current W:    | W + h (second dim):                           | gradient dW:                              
-----------------|------------------------------------------------|------------------------------------------|
[0.34,           | [0.34,                                          | [-2.5,                                    |
-1.11,          | -1.11 + 0.0001,                                  | 0.6,                                     |
0.78,           | 0.78,                                           | ? ,                                      |
0.12,           | 0.12,                                           | ? ,                                      |
0.55,           | 0.55,                                           | ? ,                                      |
2.81,           | 2.81,                                           |                                           |
-3.1,           | -3.1,                                           |                                           |
-1.5,           | -1.5,                                           |                                           |
0.33,...]       | 0.33,...]                                       | (1.25353 - 1.25347)/0.0001 = 0.6          |
loss 1.25347    | loss 1.25353                                    |                                           |

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

| 0.34,  |
| -1.11, |
| 0.78,  |
| 0.12,  |
| 0.55,  |
| 2.81,  |
| -3.1,  |
| -1.5,  |
| 0.33,..|

**W + h (third dim):**

| 0.34,   |
| -1.11,  |
| 0.78 + 0.0001, |
| 0.12,   |
| 0.55,   |
| 2.81,   |
| -3.1,   |
| -1.5,   |
| 0.33,.. |

**gradient dW:**

| [-2.5, |
| 0.6,  |
| ?,    |
| ?,    |
| ?,    |
| ?,    |
| ?,    |
| ?,    |
| ?,    |
| ?,    |

**loss 1.25347**

**loss 1.25347**
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 $W$:

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

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

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

want $\nabla_W L$
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

gradient dW:

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

\[dW = \ldots\]

(some function data and W)
Loss gradients

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

• i.e. how does the loss change 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
- Move in direction opposite to gradient:

\[ w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)}) \]
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
• Could also 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’
• Allows faster training (e.g. on GPUs), parallelization
Learning rate selection

The effects of step size (or “learning rate”)
Gradient descent in multi-layer nets

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

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

• How to update the weights at all layers?
• Answer: backpropagation of error from higher layers to lower layers
Example

- Two layer network w/ tanh at hidden layer:
  \[ h(a) \equiv \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \]

- Derivative:
  \[ h'(a) = 1 - h(a)^2 \]

- Minimize:
  \[ E_n = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2 \]

- Forward propagation:
  \[ a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \]
  \[ z_j = \tanh(a_j) \]
  \[ y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \]

output function = identity

y = prediction, t = ground truth label

Equations from Chris Bishop
Example

• Errors at output:
  \[ \delta_k = y_k - t_k \]

• Errors at hidden units:
  \[ \delta_j = (1 - z^2_j) \sum_{k=1}^{K} w_{kj} \delta_k \]

• Derivatives wrt weights:
  \[ \frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \quad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j \]

Equations from Chris Bishop
Backpropagation: Graphic example

First calculate error of output units and use this to change the top layer of weights.

\[ \delta_k = y_k - t_k \]

Update weights into \( j \)

\[ \frac{\partial E}{\partial w_{k,j}^{(2)}} = \delta_k z_j \]

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

Adapted from Ray Mooney, equations from Chris Bishop
Next calculate error for hidden units based on errors on the output units it feeds into.

\[
\delta_k = y_k - t_k
\]

\[
\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{k,j} \delta_k
\]
Finally update bottom layer of weights based on errors calculated for hidden units.

\[ \delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k \]

Update weights into \( i \)

\[ \frac{\partial E}{\partial w_{ji}^{(1)}} = \delta_j x_i \]

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

Adapted from Ray Mooney, equations from Chris Bishop
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, y = 5, z = -4 \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

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

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

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

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

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

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, \ y = 5, \ z = -4 \)

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

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

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

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

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

Want: \[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

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

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

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

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

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

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
\begin{align*}
q &= x + y & \frac{\partial q}{\partial x} &= 1, \quad \frac{\partial q}{\partial y} &= 1 \\
f &= qz & \frac{\partial f}{\partial q} &= z, \quad \frac{\partial f}{\partial z} &= q
\end{align*}
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

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

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

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

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

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

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)

Chain rule:

\[
\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y}
\]
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

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

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

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, y = 5, z = -4 \)

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

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

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:
\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x}
\]
activations

\[
\frac{\partial z}{\partial x}, \quad \frac{\partial z}{\partial y}
\]

“local gradient”

\[
f
\]

gradients
activations

\( f \)

\( \frac{\partial z}{\partial x} \)

\( \frac{\partial z}{\partial y} \)

"local gradient"

\( \frac{\partial L}{\partial z} \)

gradients

Andrej Karpathy
activations

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

"local gradient"

gradients
activations

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

\[
\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}
\]

gradients

"local gradient"
activations

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

\[ \frac{\partial z}{\partial x} \]

\[ \frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y} \]

\[ \frac{\partial z}{\partial y} \]

“local gradient”

f

\[ \frac{\partial z}{\partial z} \]

\[ \frac{\partial L}{\partial z} \]

gradients

Andrej Karpathy
Convolutional neural networks
Convolutional Neural Networks (CNN)

- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant, *more abstract* features
- Classification layer at the end

Convolutional Neural Networks (CNN)

- Feed-forward feature extraction:
  1. Convolve input with learned filters
  2. Apply non-linearity
  3. Spatial pooling (downsample)

- Supervised training of convolutional filters by back-propagating classification error

Adapted from Lana Lazebnik
1. Convolution

- Apply learned filter weights
- One feature map per filter
- Stride can be greater than 1 (faster, less memory)

Adapted from Rob Fergus
2. Non-Linearity

• Per-element (independent)

• Options:
  • Tanh
  • Sigmoid: $1/(1+\exp(-x))$
  • Rectified linear unit (ReLU)
    – Avoids saturation issues

Adapted from Rob Fergus
3. Spatial Pooling

- Sum or max over non-overlapping / overlapping regions
- Role of pooling:
  - Invariance to small transformations
  - Larger receptive fields (neurons see more of input)

Adapted from Rob Fergus
3. Spatial Pooling

- Sum or max over non-overlapping / overlapping regions
- Role of pooling:
  - Invariance to small transformations
  - Larger receptive fields (neurons see more of input)

Rob Fergus, figure from Andrej Karpathy
Convolutions: More detail

32x32x3 image

- 32x32x3 image
- Height: 32
- Width: 32
- Depth: 3
Convolutions: More detail

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolutions: More detail

Convolution Layer

32x32x3 image
5x5x3 filter $w$

1 number: the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. $5 \times 5 \times 3 = 75$-dimensional dot product + bias)

$$w^T x + b$$
Convolutions: More detail

Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Convolution Layer

- 32x32x3 image
- 5x5x3 filter
- Convolve (slide) over all spatial locations
- Consider a second, green filter

Andrej Karpathy
Convolutions: More detail

For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
Convolutions: More detail

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

CONV, ReLU e.g. 6 5x5x3 filters
ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

**Preview:**

- **First layer:** CONV, ReLU
  - Filters: 6, size 5x5x3,
  - Input: 3, output: 32

- **Second layer:** CONV, ReLU
  - Filters: 10, size 5x5x6,
  - Input: 28, output: 28

- **Third layer:** CONV, ReLU
  - Filters: 10, size 5x5x6,
  - Input: 28, output: 24

- **Continuation:** CONV, ReLU

Andrej Karpathy
Convolutions: More detail

[From recent Yann LeCun slides]

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
We call the layer convolutional because it is related to convolution of two signals:

\[
G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]
\]

Element-wise multiplication and sum of a filter and the signal (image)
Convolutions: More detail

A closer look at spatial dimensions:

- **32x32x3 image**
- **5x5x3 filter**

Convolve (slide) over all spatial locations

**activation map**
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied \textbf{with stride 2}
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2 => 3x3 output!
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied \textit{with stride 3}?
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?

doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.
Convolutions: More detail

Output size:
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3:\)
- \(\text{stride 1} \Rightarrow (7 - 3)/1 + 1 = 5\)
- \(\text{stride 2} \Rightarrow (7 - 3)/2 + 1 = 3\)
- \(\text{stride 3} \Rightarrow (7 - 3)/3 + 1 = 2.33\)
Convolutions: More detail

In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

(recall:)
\[(N - F) / \text{stride} + 1\]
In practice: Common to zero pad the border

E.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

7x7 output!
In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

**7x7 output!**

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
    F = 5 => zero pad with 2
    F = 7 => zero pad with 3

\[(N + 2*padding - F) / stride + 1\]
Convolutions: More detail

Examples time:

Input volume: $32 \times 32 \times 3$
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Convolutions: More detail

Examples time:

Input volume: $32 \times 32 \times 3$
10 5x5 filters with stride 1, pad 2

Output volume size:
$\frac{32+2*2-5}{1}+1 = 32$ spatially, so $32 \times 32 \times 10$
Convolutions: More detail

Examples time:

Input volume: \textbf{32x32x3}
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
Convolutions: More detail

Examples time:

Input volume: \(32 \times 32 \times 3\)
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has \(5 \times 5 \times 3 + 1 = 76\) params (+1 for bias)
\(76 \times 10 = 760\)
Convolutions: More detail
A Common Architecture: AlexNet

Figure from http://www.mdpi.com/2072-4292/7/11/14680/htm
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013

->

7.3% top 5 error
Case Study: GoogLeNet

Inception module

ILSVRC 2014 winner (6.7% top 5 error)
Case Study: ResNet

[He et al., 2015]
ILSVRC 2015 winner (3.6% top 5 error)

MSRA @ ILSVRC & COCO 2015 Competitions

• 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

*improvements are relative numbers

Slide from Kaiming He’s recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w
Case Study: ResNet

Revolution of Depth

ImageNet Classification top-5 error (%)


(slide from Kaiming He’s recent presentation)
Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)

2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet!
(even though it has 8x more layers)

Revolution of Depth

AlexNet, 8 layers (ILSVRC 2012)

VGG, 19 layers (ILSVRC 2014)

ResNet, 152 layers (ILSVRC 2015)

(slide from Kaiming He’s recent presentation)
Practical matters
Comments on training algorithm

• Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
• However, in practice, does converge to low error for many large networks on real data.
• Thousands of epochs (epoch = network sees all training data once) may be required, hours or days to train.
• To avoid local-minima problems, run several trials starting with different random weights \( (\text{random restarts}) \), and take results of trial with lowest training set error.
• May be hard to set learning rate and to select number of hidden units and layers.
• Neural networks had fallen out of fashion in 90s, early 2000s; back with a new name and significantly improved performance (deep networks trained with dropout and lots of data).
Over-training prevention

- Running too many epochs can result in over-fitting.

- Keep a hold-out validation set and test accuracy on it after every epoch. Stop training when additional epochs actually increase validation error.

Adapted from Ray Mooney
Training: Best practices

- Use mini-batch
- Use regularization
- Use gradient checks
- Use cross-validation for your parameters
- Use RELU or leaky RELU or ELU, don’t use sigmoid
- Center (subtract mean from) your data
- To initialize, use “Xavier initialization”
- Learning rate: too high? Too low?
Regularization: Dropout

- Randomly turn off some neurons
- Allows individual neurons to independently be responsible for performance

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

Adapted from Jia-bin Huang
Data Augmentation (Jittering)

Create *virtual* training samples

- Horizontal flip
- Random crop
- Color casting
- Geometric distortion

Deep Image [Wu et al. 2015]
Transfer Learning

“You need a lot of data if you want to train use CNNs”

Andrej Karpathy
Transfer Learning with CNNs

• The more weights you need to learn, the more data you need
• That’s why with a deeper network, you need more data for training than for a shallower network
• One possible solution:

Set these to the already learned weights from another network  
Learn these on your own task
Transfer Learning with CNNs

Source: classification on ImageNet

1. Train on ImageNet

2. Small dataset:
   - Freeze these
   - Train this

3. Medium dataset: finetuning
   - more data = retrain more of the network (or all of it)
   - Freeze these
   - Train this

Another option: use network as feature extractor, train SVM on extracted features for target task

Adapted from Andrej Karpathy
## Transfer Learning with CNNs

<table>
<thead>
<tr>
<th>Dataset Description</th>
<th>Action 1</th>
<th>Action 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very similar dataset</td>
<td>Use linear classifier on top layer</td>
<td>You’re in trouble… Try linear classifier from different stages</td>
</tr>
<tr>
<td>Very little data</td>
<td>Finetune a few layers</td>
<td>Finetune a larger number of layers</td>
</tr>
<tr>
<td>Quite a lot of data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- More generic
- More specific

- Andrej Karpathy
Pre-training on ImageNet

- Have a source domain and target domain
- Train a network to classify ImageNet classes
  - Coarse classes and ones with fine distinctions (dog breeds)
- Remove last layers and train layers to replace them, that predict target classes

Oquab et al., “Learning and Transferring Mid-Level Image Representations…”, CVPR 2014
Transfer learning with CNNs is pervasive...

Object Detection
Ren et al., “Faster R-CNN“, NIPS 2015

Image Captioning

CNN pretrained on ImageNet
Semantic segmentation

1. Extract patch
2. Run through a CNN
3. Classify center pixel
4. Repeat for every pixel

Andrej Karpathy
Photographer identification

Who took this photograph?

- Deep net features achieve 74% accuracy
  - Chance is less than 3%, human performance is 47%
- Method learns what proto-objects + scenes authors shoot

Thomas and Kovashka, CVPR 2016
Analysis of pre-training on ImageNet

- Target tasks:
  - object detection and action recognition on PASCAL
  - scene recognition on SUN
- Pre-training with 500 images per class is about as good as pre-training with 1000
- Pre-training with 127 classes is about as good as pre-training with 1000
- Pre-training with (fewer classes, more images per class) > (more classes, fewer images)
- Small drop in if classes with fine-grained distinctions removed from pre-training set

Packages

Caffe and Caffe Model Zoo
Torch
Theano with Keras/Lasagne
MatConvNet
TensorFlow
Learning Resources

http://deeplearning.net/
http://cs231n.stanford.edu
Understanding CNNs
Recall: Biological analog

Hubel and Weisel’s architecture

Multi-layer neural network

Adapted from Jia-bin Huang
Layer 1

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Layer 2

- Activations projected down to pixel level via deconvolution
- Patches from validation images that give maximal activation of a given feature map

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Layer 3

- 9 Patches
Layer 4 and 5

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Occlusion experiments

(a) Input Image

True Label: Pomeranian

(d) Classifier, probability of correct class

(as a function of the position of the square of zeros in the original image)

True Label: Car Wheel

True Label: Afghan Hound

[Zeiler & Fergus 2014]
Occlusion experiments

(a) Input Image
(b) Classifier, probability of correct class

(True Label: Pomeranian)

(True Label: Car Wheel)

(True Label: Afghan Hound)

(as a function of the position of the square of zeros in the original image)

[Zeiler & Fergus 2014]

Andrej Karpathy
What image maximizes a class score?

Repeat:
1. Forward an image
2. Set activations in layer of interest to all zero, except for a 1.0 for a neuron of interest
3. Backprop to image
4. Do an “image update”
What image maximizes a class score?

[Understanding Neural Networks Through Deep Visualization, Yosinski et al., 2015]
http://yosinski.com/deepvis
What image maximizes a class score?
## Breaking CNNs

<table>
<thead>
<tr>
<th>correct</th>
<th>+distort</th>
<th>ostrich</th>
<th>correct</th>
<th>+distort</th>
<th>ostrich</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Correct Image" /></td>
<td><img src="image2.png" alt="Distorted Image" /></td>
<td><img src="image3.png" alt="Ostrich Image" /></td>
<td><img src="image4.png" alt="Correct Image" /></td>
<td><img src="image5.png" alt="Distorted Image" /></td>
<td><img src="image6.png" alt="Ostrich Image" /></td>
</tr>
</tbody>
</table>

Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

Intriguing properties of neural networks [Szegedy ICLR 2014]
Breaking CNNs

Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images [Nguyen et al. CVPR 2015]
Fooling a linear classifier

To fool a linear classifier, add a small multiple of the weight vector to the training example:

\[ \mathbf{x} \leftarrow \mathbf{x} + \alpha \mathbf{w} \]

http://karpathy.github.io/2015/03/30/breaking-convnets/
Summary

• We use deep neural networks because of their strong performance in practice

• Convolutional neural network (CNN)
  • Convolution, nonlinearity, max pooling

• Training deep neural nets
  • We need an objective function that measures and guides us towards good performance
  • We need a way to minimize the loss function: stochastic gradient descent
  • We need backpropagation to propagate error towards all layers and change weights at those layers

• Practices for preventing overfitting
  • Dropout; data augmentation; transfer learning