Convolutional neural networks

Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1
Announcements

• Monday is a Wellness Day
• Project 5: *New* To be assigned Wednesday, April 28, due Tuesday, May 11
• By a large majority, respondents preferred original final exam time: assigned Wednesday, May 12, 2021; due Monday, May 17, 2021
• Sample final exam to be released soon
Readings

• Neural networks

• Convolutional neural networks
Image Classification: a core task in computer vision

- Assume given set of discrete labels, e.g. 
  \{\text{cat, dog, cow, apple, tomato, truck, ...}\}

\[
\begin{align*}
    f(\text{apple}) &= \text{“apple”} \\
    f(\text{tomato}) &= \text{“tomato”} \\
    f(\text{cow}) &= \text{“cow”}
\end{align*}
\]
Recap: linear classification

• Have score function and loss function
  – Score function maps an input data instance (e.g., an image) to a vector of scores, one for each category
  – Last time, our score function is based on linear classifier

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

\( f \): score function
\( x \): input instance
\( W, b \): parameters of a linear (actually affine) function

• Find \( W \) and \( b \) to minimize a loss, e.g. cross-entropy loss

\[
L = \frac{1}{N} \sum_i - \log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right)
\]
Linear classifiers separate features space into half-spaces

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

(Before) Linear score function: \( f = Wx \)
Neural networks

**Before** Linear score function: $f = Wx$

**Now** 2-layer Neural Network: $f = W_2 \max(0, W_1 x)$
Neural networks

(Before) Linear score function:
\[ f = Wx \]

(Now) 2-layer Neural Network
\[ f = W_2 \max(0, W_1 x) \]

\[ W_1 \quad (100 \times 3072 \text{ matrix}) \]
\[ h \]
\[ W_2 \quad (10 \times 100 \text{ matrix}) \]

100D intermediate vector
Neural networks

(Before) Linear score function: \( f = Wx \)

(Now) 2-layer Neural Network
\[
\begin{align*}
    &f = W_2 \max(0, W_1 x) \\
    &x \rightarrow W_1 \rightarrow h \rightarrow W_2 \rightarrow s \\
    &3072 \rightarrow 100 \rightarrow 10
\end{align*}
\]

- Total number of weights to learn:
  \[
  3,072 \times 100 + 100 \times 10 = 308,200
  \]
Recap: linear classification

- Have score function and loss function
  - Score function maps an input data instance (e.g., an image) to a vector of scores, one for each category
  - Last time, our score function is based on linear classifier
    \[ f(x, W) = Wx + b \]
    - \( f \): score function
    - \( x \): input instance
    - \( W, b \): parameters of a linear (actually affine) function

- Find \( W \) and \( b \) to minimize a loss, e.g. cross-entropy loss

\[
L = \frac{1}{N} \sum_{i} -\log \left( \frac{e^{f_{y_i}}}{\sum_{j} e^{f_{j}}} \right)
\]
Neural networks

(Before) Linear score function:  
\[ f = W x \]

(Now) 2-layer Neural Network or 3-layer Neural Network  
\[ f = W_2 \max(0, W_1 x) \]
\[ f = W_3 \max(0, W_2 \max(0, W_1 x)) \]

also called “Multi-Layer Perceptrons” (MLPs)
Neural networks

• Very coarse generalization of neural networks:
  – Linear functions chained together and separated by non-linearities (activation functions), e.g. "max"

\[ f = W_3 \max(0, W_2 \max(0, W_1 x)) \]

  – Why separate linear functions with non-linear functions?
  – Very roughly inspired by real neurons
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**

\[
\begin{cases} 
  x & x \geq 0 \\
  \alpha(e^x - 1) & x < 0 
\end{cases}
\]
Neural network architecture

- Computation graph for a 2-layer neural network
Deep networks typically have many layers and potentially millions of parameters.
Deep neural network

• *Inception* network (Szegedy et al, 2015)
• 22 layers
• Just like a linear classifier – but in this case, just one layer of a larger network
Example feed-forward computation of a neural network

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```
Summary

- We arrange neurons into fully-connected layers
- The abstraction of a \textit{layer} has the nice property that it allows us to use efficient vectorized code (e.g. matrix multiplies)
- Neural networks are not really \textit{neural}
Optimizing parameters with gradient descent

• How do we find the best $W$ and $b$ parameters?
• In general: *gradient descent*
  1. Start with a guess of a good $W$ and $b$ (or randomly initialize them)
  2. Compute the loss function for this initial guess and the *gradient* of the loss function
  3. Step some distance in the negative gradient direction (direction of steepest descent)
  4. Repeat steps 2 & 3
• Note: efficiently performing step 2 for deep networks is called *backpropagation*
Gradient descent: walk in the direction opposite gradient

• Q: How far?
• A: Step size: learning rate
• Too big: will miss the minimum
• Too small: slow convergence
2D example of gradient descent

- In reality, in deep learning we are optimizing a highly complex loss function with millions of variables (or more)
- More on this later...

2D example: TensorFlow Playground

https://playground.tensorflow.org
Questions?
Convolutional neural networks

Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1
A bit of history...

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

recognized letters of the alphabet

**update rule:**

\[ w_i(t + 1) = w_i(t) + \alpha (d_j - y_j(t))x_{j,i} \]

Frank Rosenblatt, ~1957: Perceptron
A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

A bit of history:
Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]
First strong results

**Acoustic Modeling using Deep Belief Networks**
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010

**Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition**
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

**Imagenet classification with deep convolutional neural networks**

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

“AlexNet”
Fast-forward to today: ConvNets are everywhere

Fast-forward to today: ConvNets are everywhere

Detection

Segmentation

Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]


[Farabet et al., 2012]
Fast-forward to today: ConvNets are everywhere

self-driving cars

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.
Fast-forward to today: ConvNets are everywhere

[Toshev, Szegedy 2014]

[Guo et al. 2014]
Fast-forward to today: ConvNets are everywhere

[Levy et al. 2016]

Figure copyright Levy et al. 2016. Reproduced with permission.

[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/ESO/NA, public domain by NASA, and public domain.

[Serbanet et al. 2011]

[Caressan et al.]
A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

All images are CC0 Public domain:

Captions generated by Justin Johnson using LucidText2
DALL·E: Creating Images from Text, OpenAI
https://openai.com/blog/dall-e/

**Original Images:** In CC0 public domain. 
*Starry Night* and *Tree Roots* by Van Gogh are in the public domain. 
*Polish Images* is in the public domain.

Stylized images copyright Justin Johnson, 2017; reproduced with permission.

Gatys et al, “Controlling Perceptual Factors in Neural Style Transfer”, CVPR 2017
Convolutional neural networks

- Version of deep neural networks designed for signals
  - 1D signals (e.g., speech waveforms)
  - 2D signals (e.g., images)
Motivation – Feature Learning
Life Before Deep Learning

Input Pixels   Extract Hand-Crafted 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 $x = \text{pixels}$)
Linearly separable classes
Aside: Image Features

\[ f(x) = Wx \]
Aside: Image Features

\[ f(x) = Wx \]

Feature Representation

Class scores
Image Features: Motivation

Cannot separate red and blue points with linear classifier
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
Image features vs ConvNets

Feature Extraction

10 numbers giving scores for classes

Training

10 numbers giving scores for classes

Training
Last layer of most CNNs is a linear classifier

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

This piece is just a linear classifier

Key: perform enough processing so that by the time you get to the end of the network, the classes are linearly separable
Visualizing AlexNet in 2D with t-SNE

Convolutional neural networks

• Layer types:
  – Fully-connected layer
  – Convolutional layer
  – Pooling layer
Fully Connected Layer

32x32x3 image \rightarrow\text{stretch to } 3072 \times 1

input

\begin{array}{c}
1 \\
3072
\end{array}

\rightarrow

Wx

\begin{array}{c}
10 \times 3072 \\
\text{weights}
\end{array}

\rightarrow

activation

\begin{array}{c}
1 \\
10
\end{array}
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Same as a linear classifier!
Convolution Layer

32x32x3 image -> preserve spatial structure
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Number of weights: \( 5 \times 5 \times 3 + 1 = 76 \) (vs. 3072 for a fully-connected layer) (+1 for bias)

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
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. $5*5*3 = 75$-dimensional dot product + bias)

$w^T x + b$
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Consider a second, green filter.

32x32x3 image
5x5x3 filter

Convolve (slide) over all spatial locations

Activation maps
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!

(total number of parameters: $6 \times (75 + 1) = 456$)
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions.

CONV, ReLU

32 → 28

e.g. 6 5x5x3 filters
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

Preview

Low-level features → Mid-level features → High-level features → Linearily separable classifier

VGG-16 Conv1_1 → VGG-16 Conv3_2 → VGG-16 Conv5_3
A closer look at spatial dimensions:

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
applied with stride 2
=> 3x3 output!
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied **with stride 3**?
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.
Output size:
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3\):

- \(\text{stride } 1 \Rightarrow (7 - 3)/1 + 1 = 5\)
- \(\text{stride } 2 \Rightarrow (7 - 3)/2 + 1 = 3\)
- \(\text{stride } 3 \Rightarrow (7 - 3)/3 + 1 = 2.33\)

\(\backslash\)
In practice: Common to zero pad the border

0 0 0 0 0 0
0
0
0
0

Example 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

\[ \begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & & & & & \\
0 & & & & & \\
0 & & & & & \\
0 & & & & & \\
0 & & & & & \\
\end{array} \]

- 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>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>0</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>
</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
Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn’t work well.
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Examples time:

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

Output volume size: \\
\((32+2*2-5)/1+1 = 32\) spatially, so \\
\textbf{32x32x10}
Examples time:

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

Number of parameters in this layer?
Examples time:

Input volume: 32x32x3
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)
\[76 \times 10 = 760\]
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Convolutional layer—properties

- Small number of parameters to learn compared to a fully connected layer
- Preserves spatial structure—output of a convolutional layer is shaped like an image
- **Translation equivariant:** passing a translated image through a convolutional layer is (almost) equivalent to translating the convolution output (but be careful of image boundaries)
Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently:
MAX POOLING

Single depth slice

max pool with 2x2 filters and stride 2

6 8

3 4
Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks
[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
AlexNet
“AlexNet” [Krizhevsky et al. NIPS 2012]

“GoogLeNet” [Szegedy et al. CVPR 2015]

“VGG Net” [Simonyan & Zisserman, ICLR 2015]

“ResNet” [He et al. CVPR 2016]
Big picture

• A convolutional neural network can be thought of as a function from images to class scores
  – With millions of adjustable weights...
  – ... leading to a very non-linear mapping from images to features / class scores.
  – We will set these weights based on classification accuracy on training data...
  – ... and hopefully our network will generalize to new images at test time
Data is key—enter ImageNet

• ImageNet (and the ImageNet Large-Scale Visual Recognition Challenge, aka ILSVRC) has been key to training deep learning methods

• ILSVRC: 1,000 object categories, each with ~700-1300 training images. Test set has 100 images per categories (100,000 total).

• Standard ILSVRC error metric: top-5 error
  – if the correct answer for a given test image is in the top 5 categories, your answer is judged to be correct
Performance improvements on ILSVRC

- ImageNet Large-Scale Visual Recognition Challenge
- Held from 2011-2017
- 1000 categories, 1000 training images per category
- Test performance on held-out test set of images

AlexNet
Questions?