Unstructured Data Analysis

Lecture 12: Image analysis with convolutional neural nets

George Chen
Architecting Neural Nets

• Basic building block that is often repeated: linear layer followed by nonlinear activation
  • Without nonlinear activation, two consecutive linear layers is mathematically equivalent to having a single linear layer!
• How to select # of nodes in a layer, or # of layers?
  • These are hyperparameters! Infinite possibilities!
  • Can choose between different options using hyperparameter selection strategy from earlier lectures
    • Very expensive in practice!
      (Active area of research: neural architecture search)
• Much more common in practice: modify existing architectures that are known to work well
  (e.g., ResNet for image classification/object recognition)
PyTorch GitHub Has Lots of Examples

A repository showcasing examples of using PyTorch

- Image classification (MNIST) using Convnets
- Word level Language Modeling using LSTM RNNs
- Training Imagenet Classifiers with Residual Networks
- Generative Adversarial Networks (DCGAN)
- Variational Auto-Encoders
- Superresolution using an efficient sub-pixel convolutional neural network
- Hogwild training of shared ConvNets across multiple processes on MNIST
- Training a CartPole to balance in OpenAI Gym with actor-critic
- Natural Language Inference (SNLI) with GloVe vectors, LSTMs, and torchtext
- Time sequence prediction - use an LSTM to learn Sine waves
- Implement the Neural Style Transfer algorithm on images
- Several examples illustrating the C++ Frontend

Additionally, a list of good examples hosted in their own repositories:

- Neural Machine Translation using sequence-to-sequence RNN with attention (OpenNMT)
Find a Massive Collection of Models at the Model Zoo

Model Zoo
Discover open source deep learning code and pretrained models.

Browse Frameworks  Browse Categories
Deep Learning

- Inspired by biological neural nets *but otherwise not the same at all* (biological neural nets do *not* work like deep nets)
- Learns a layered representation
  - Tries to get rid of manual feature engineering
  - Need to design constraints for what features are learned to account for structure in data (e.g., images, text, …)
Learning a neural net amounts to curve fitting

We’re just estimating a function
Neural Net as Function Approximation

Given **input**, learn a **computer program** that computes **output**

Multinomial logistic regression:

```python
def f(input):
    output = softmax(np.dot(input, W) + b)
    return output
```

the only things that we are learning (we fix their dimensions in advance)

We are fixing what the function `f` looks like in code and are only adjusting `W` and `b`!!!
Neural Net as Function Approximation

Given input, learn a computer program that computes output

Multinomial logistic regression:

\[
\text{output} = \text{softmax}(\text{np.dot}(\text{input}, W) + b)
\]

Multilayer perceptron:

\[
\text{intermediate} = \text{relu}(\text{np.dot}(\text{input}, W1) + b1)
\]
\[
\text{output} = \text{softmax}(\text{np.dot}(\text{intermediate}, W2) + b2)
\]

Learning a neural net: learning a simple computer program that maps inputs (raw feature vectors) to outputs (predictions)
Complexity of a Neural Net?

- Increasing number of layers (depth) makes neural net more “complex”
  - Learn computer program that has more lines of code
  - Sometimes, more parameters may be needed
    - If so, more training data may be needed

Earlier: multinomial logistic regression had fewer parameters than multilayer perceptron example

Upcoming: we’ll see examples of deep nets with fewer parameters than “shallower” nets
Image analysis with Convolutional Neural Nets (CNNs, also called convnets)
Convolution

filter

Slide by Phillip Isola
Convolution

Input image

Filter (also called “kernel”)
Convolution

Input image

Filter (also called “kernel”)
Convolution

Take dot product!

Input image

Output image
Convolution

Take dot product!

<table>
<thead>
<tr>
<th></th>
<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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Input image

Output image

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Take dot product!
Convolution

Take dot product!

Input image

Output image

0 0 0 0 0 0 0 0
0 0 0 1 0 1 1 0
0 1 1 0 1 1 1 1
0 1 1 1 1 1 1 0
0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0

0 1 1

Convolution

Take dot product!

Input image

Output image

0 0 0 0
0 0 1 0
0 1 1 0
0 1 1 0
0 1 1 0
0 1 1 0
0 0 1 0
0 0 0 0

0 1 1 1

0 1 1 1
0 1 1 1
0 1 1 1
0 0 1 1
0 0 0 0
Convolution

Take dot product!

Input image

Output image
Convolution

Take dot product!

Input image

Output image
Convolution

Take dot product!

Input image

Output image
Convolution

Input image

Output image

Note: output image is smaller than input image
If you want output size to be same as input, pad 0’s to input
**Convolution**

Input image

Note: output image is smaller than input image

If you want output size to be same as input, pad 0’s to input

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 0 0 0
0 0 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 0 0 0 0
0 0 1 1 1 1 1 1 0 0
```

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 0 0 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0
```

```
0 0 0
0 1 0
0 0 0
```

```
0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0
0 1 1 1 1 1 1 0 0 0
0 1 1 1 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```
Convolution

Input image

Output image
Convolution

Input image

Output image
Convolution

Input image

Output image

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \ast \begin{bmatrix}
-1 & -1 & -1 \\
2 & 2 & 2 \\
-1 & -1 & -1
\end{bmatrix} = \begin{bmatrix}
0 & 1 & 3 & 1 & 0 \\
1 & 1 & 1 & 3 & 3 \\
0 & 0 & -2 & -4 & -4 \\
1 & 1 & 1 & 3 & 3 \\
0 & 1 & 3 & 1 & 0
\end{bmatrix}$$
Convolution

Very commonly used for:

• Blurring an image

\[
\begin{array}{ccc}
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9
\end{array}
\]

* =

\[
\begin{array}{ccc}
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9
\end{array}
\]

• Finding edges

\[
\begin{array}{ccc}
-1 & -1 & -1 \\
2 & 2 & 2 \\
-1 & -1 & -1
\end{array}
\]

* =

\[
\begin{array}{ccc}
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9 \\
1/9 & 1/9 & 1/9
\end{array}
\]

(this example finds horizontal edges)

Images from: http://aishack.in/tutorials/image-convolution-examples/
Convolution Layer

Images from: http://aishack.in/tutorials/image-convolution-examples/

Conv2d layer

Convolve with each filter

1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9

-1 -1 -1
2 2 2
-1 -1 -1

0 -1 0
-1 4 -1
0 -1 0

Filters & biases (1 bias number per filter) are unknown and are learned!

Add bias

Apply activation

Activation layer (such as ReLU)
Convolution Layer

Images from: http://aishack.in/tutorials/image-convolution-examples/
Convolution Layer

Input dimensions: 1 (# channels), height, width

Conv2d
(3 kernels, each size 3x3), ReLU activation

Stack output images into a single “output feature map”

dimensions: 3, height-2, width-2

Images from: http://aishack.in/tutorials/image-convolution-examples/
**Convolution Layer**

Conv2d

- $(k$ kernels each size $3 \times 3$), ReLU activation

Input dimensions:
- 1 (# channels), height, width

Conv2d output dimensions:
- $k$, height-2, width-2

Stack output images into a single “output feature map”

Images from: http://aishack.in/tutorials/image-convolution-examples/
Convolution Layer

Input

dimensions:
- \(d\) (# channels)
- height, width

Conv2d
- \(k\) kernels
- each size \(d \times 3 \times 3\)
- ReLU activation

Stack output images into a single “output feature map”

dimensions:
- \(k\)
- height-2, width-2

Images from: http://aishack.in/tutorials/image-convolution-examples/
Convolution Layer

Input dimensions: 
- $d$ (# channels)
- height, width

Conv2d
- $(k$ kernels each size $d \times 3 \times 3$), ReLU activation

Stack output images into a single “output feature map”

- dimensions: 
  - $k$, height-2, width-2

Each filter:
- $d$  
  - image height
  - image width
  - $d$
Pooling

- Aggregate local information ("pool" together information)

- Produces a smaller image
  (each resulting pixel captures some "global" information)

- If "object" in input image shifts, want output to stay the same
Max Pooling

Convolution layer (1 filter, for simplicity no bias)

Input

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
-1 & -1 & -1 \\
2 & 2 & 2 \\
-1 & -1 & -1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 1 & 3 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 3 & 3 & 0 & 0 & 0 \\
0 & 1 & 3 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

*
Max Pooling

Convolution layer (1 filter, for simplicity no bias)

Input

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
-1 & -1 & -1 \\
2 & 2 & 2 \\
-1 & -1 & -1 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
0 & 1 & 3 & 1 & 0 \\
1 & 1 & 1 & 3 & 3 \\
0 & 0 & -2 & -4 & -4 \\
1 & 1 & 1 & 3 & 3 \\
0 & 1 & 3 & 1 & 0 \\
\end{array}
\]

Output after ReLU

Output after max pooling
Max Pooling

Convolution layer (1 filter, for simplicity no bias)

Input:

<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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Output image after ReLU:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>3</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Output after max pooling:

1
Max Pooling

Convolution layer (1 filter, for simplicity no bias)

Input

Output after ReLU

Output after max pooling
Max Pooling
Convolution layer (1 filter, for simplicity no bias)

<table>
<thead>
<tr>
<th></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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Input

\[
\begin{array}{ccc}
-1 & -1 & -1 \\
2 & 2 & 2 \\
-1 & -1 & -1 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 3 & 1 \\
1 & 1 & 1 & 3 \\
0 & 0 & -2 & -4 \\
1 & 1 & 1 & 3 \\
0 & 1 & 3 & 1 \\
\end{array}
\]

Output image after ReLU

\[
\begin{array}{cccc}
0 & 1 & 3 & 1 \\
1 & 1 & 1 & 3 \\
0 & 0 & 0 & 0 \\
1 & 1 & 1 & 3 \\
0 & 1 & 3 & 1 \\
\end{array}
\]

Output after max pooling

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Max Pooling

Convolution layer (1 filter, for simplicity no bias)

\[
\begin{align*}
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} & \times & \\
\begin{bmatrix}
-1 & -1 & -1 \\
2 & 2 & 2 \\
-1 & -1 & -1
\end{bmatrix} & = & \\
\begin{bmatrix}
0 & 1 & 3 & 1 & 0 \\
1 & 1 & 1 & 3 & 3 \\
0 & 0 & -2 & -4 & -4 \\
1 & 1 & 3 & 3 \\
0 & 1 & 3 & 1 & 0
\end{bmatrix} & \times & \\
\begin{bmatrix}
0 & 1 & 3 & 1 & 0 \\
1 & 1 & 3 & 3 \\
0 & 0 & 0 & 0 & 0 \\
1 & 1 & 3 & 3 \\
0 & 1 & 3 & 1 & 0
\end{bmatrix}
\end{align*}
\]

Output image after ReLU

Output after max pooling
Max Pooling

Convolution layer (1 filter, for simplicity no bias)

What numbers were involved in computing this 1?
In this example: 1 pixel in max pooling output captures information from 16 input pixels!
Example: applying max pooling again results in a single pixel that captures info from entire input image!
Max Pooling and (Slight) Shift Invariance

Small shift of “object” (e.g., a detected edge) in input image results in same output
Max Pooling and (Slight) Shift Invariance

Bigger shift in input can still change output
Basic Building Block of CNNs

Images from: http://aishack.in/tutorials/image-convolution-examples/
Handwritten Digit Recognition

Training label: 6

Input

Flatten (512 nodes), ReLU

Linear (10 nodes), Softmax

Loss → error

Categorical cross entropy

Important: in lecture, I will sometimes use this notation instead
Handwritten Digit Recognition

Training label: 6

Input → Conv2d, ReLU → Max Pool 2d → Flatten → Linear (10 nodes), Softmax → Loss → error

Categorical cross entropy
Handwritten Digit Recognition

Training label: 6

Input

Conv2d, ReLU, Max Pool 2d

Conv2d, ReLU, Max Pool 2d

Flatten

Linear (10 nodes), Softmax

Loss → error

Categorical cross entropy

extract low-level visual features & aggregate

extract higher-level visual features & aggregate

non-vision-specific classifier
CNNs

Demo
CNNs

- Learn convolution filters for extracting simple features

- Max pooling produces a *smaller* summary output and is somewhat invariant to small shifts in input “objects”
  - For examples where max pooling fails to achieve this and for a better way to do pooling, see Richard Zhang’s fix for max pooling linked on the course webpage

- Repeat convolution→activation→pooling to learn increasingly higher-level features
Visualizing What a Deep Net Learned

- Very straight-forward for CNNs
  - Plot filter outputs at different layers
  - Plot regions that maximally activate an output neuron

Images: Francois Chollet’s “Deep Learning with Python” Chapter 5
Example: Wolves vs Huskies

(a) Husky classified as wolf  (b) Explanation

Turns out the deep net learned that wolves are wolves because of snow…

➔ visualization is crucial!