Neural Networks, Architectures

Junxian He
Apr 17, 2024
Recap: Multilayer Perceptron Neural Networks (MLP)
Convolu	ional Neural Networks
Convolution is template matching

- with a sliding window
- abstract templates
- similarity measured by dot product
- stronger activation, better matching
Convolution: a 1-D example

![Convolution Diagram]

- Input sequence: 1 1 1 1
- Filter sequence: -1 0 1
- Output sequence: 2 2 2 2
Convolution: a 1-D example

- sliding window
- dot product
Convolution: a 1-D example

- sliding window
- dot product
Convolution: a 1-D example

- sliding window
- dot product
Convolution: a 1-D example

- Sliding window
- Dot product
Convolution: a 2-D example

input

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

filter

\[
\begin{array}{ccc}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{array}
\]

output
Convolution: a 2-D example

input

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

filter

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

output

-3

- sliding window
- dot product
Convolution: a 2-D example

**Input**

```
0 1 2 1 0 0 0 0 0
0 0 0 0 0 1 1 0
0 1 1 1 1 1 1 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 1 1 1 0 0 0
0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0
```

**Filter**

```
1 2 1
0 0 0
-1 -2 -1
```

**Output**

```
-3 -4
```

- Sliding window
- Dot product
**Convolution: a 2-D example**

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

- Sliding window
- Dot product
Convolution: a 2-D example

- Sliding window
- Dot product
Convolution: a 2-D example

\[ y[n, m] = \sum_{i=-r}^{r} \sum_{j=-r}^{r} w[i, j] x[n + i, m + j] \]

- **\( y[n, m] \)**: output map
- **\( i = -r \) to \( i = r \)**: coordinates in a local window
- **\( j = -r \) to \( j = r \)**: filter weights
- **\( w[i, j] \)**: input map
- **\( x[n + i, m + j] \)**: \( r \): kernel radius
  - kernel size = \( 2r + 1 \)
Convolutions: 2-D

* filter
\[
\begin{array}{ccc}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{array}
\]
= 

16
Convolution: Multi-channel outputs

Understanding the filter/kernel as feature extractors
Convolutions: Multi-channel inputs

Like (R, G, B) color notations have three features
Convolution: tensor views

- Tensor: high-dimension array
- Feature maps
  - 3-D tensor: C × H × W
  - C: channels
  - H: height
  - W: width
Convolutions:

The same filter tensor applies to different locations

- Tensor: high-dimension array
- Feature maps
  - 3-D tensor: $C \times H \times W$
  - $C$: channels
  - $H$: height
  - $W$: width
- Filters
  - 4-D tensor: $C_o \times C_i \times K_h \times K_w$
  - $C_o$: output channels
  - $C_i$: input channels
  - $K_h$, $K_w$: filter height, width
Convolution: # parameters and # operations

• # parameters
  • weights: $C_o \times C_i \times K_h \times K_w$
  • bias: $C_o$

• # floating-point operations (FLOPs)
  • # params $\times H_o \times W_o$
Convolution: padding

input: $H \times W = 8 \times 8$

output: $H \times W = 6 \times 6$

$H_{out} = H_{in} - K_h + 1$
**Convolution: padding**

- **Input:** $8 \times 8$, + pad
- **Filter:**
- **Output:** $H \times W = 8 \times 8$

- $pad = \lfloor \text{kernel\_size} / 2 \rfloor$
- maintains feature map size

- $H_{out} = H_{in} + 2pad_h - K_h + 1$
Convolution: stride

stride = 2
Convolution: stride

Stride = 2
Convolution: stride

input: $H \times W = 8 \times 8$

output: $H \times W = 4 \times 4$

stride = 2

- reduces feature map size
- compress and abstract

$H_{out} = \lceil (H_{in} + 2\text{pad}_h - K_h) / \text{str} \rceil + 1$
Convolution: translation-invariance

- Process each window in the same way

apply the same weights regardless of the window location
Deep Convolutional Networks

[Convolution + Nonlinear activation] + Pooling

LeNet – tanh activation

\[ f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \]
Deep Convolutional Networks

[Convolution + Nonlinear activation] + Pooling

Convolution

Pooling
Pooling = Down-sampling

Max pooling
Deep Convolutional Networks

[Image of a deep convolutional network diagram with layers labeled C1: 6x28x28, S2: 6x14x14, C3: 16x10x10, S4: 16x5x5, C5: 120, F6: 84, and output 10.]

60,000 original dataset
Test error: 0.95%

MNIST

[1] LeNet 5, LeCun et al. 1998
Misclassified examples on MNIST

True label -> Predicted label
Alex Net

ImageNet

- 15M images
- 22K categories
- Images collected from Web
- Human labelers (Amazon’s Mechanical Turk crowd-sourcing)
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
  - 1K categories
  - 1.2M training images (~1000 per category)
  - 50,000 validation images
  - 150,000 testing images

- RGB images
- Variable-resolution, but this architecture scales them to 256x256 size
### ImageNet Results

<table>
<thead>
<tr>
<th>mite</th>
<th>container ship</th>
<th>motor scooter</th>
<th>leopard</th>
</tr>
</thead>
<tbody>
<tr>
<td>black widow</td>
<td>container ship</td>
<td>motor scooter</td>
<td>jaguar</td>
</tr>
<tr>
<td>cockroach</td>
<td>lifeboat</td>
<td>go-kart</td>
<td>cheetah</td>
</tr>
<tr>
<td>tick</td>
<td>amphibian</td>
<td>moped</td>
<td>snow leopard</td>
</tr>
<tr>
<td>starfish</td>
<td>fireboat</td>
<td>bumper car</td>
<td>Egyptian cat</td>
</tr>
<tr>
<td></td>
<td>drilling platform</td>
<td>golfcart</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>grille</th>
<th>mushroom</th>
<th>cherry</th>
<th>Madagascar cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>convertible</td>
<td>agaric</td>
<td>dalmatian</td>
<td>squirrel monkey</td>
</tr>
<tr>
<td>grille</td>
<td>mushroom</td>
<td>grape</td>
<td>spider monkey</td>
</tr>
<tr>
<td>grille</td>
<td>mushroom</td>
<td>elderberry</td>
<td>titi</td>
</tr>
<tr>
<td>grille</td>
<td>mushroom</td>
<td>gill fungus</td>
<td>indri</td>
</tr>
<tr>
<td>pick up</td>
<td>agaric</td>
<td>gill fungus</td>
<td></td>
</tr>
<tr>
<td>beach wagon</td>
<td>grape</td>
<td>gill fungus</td>
<td></td>
</tr>
<tr>
<td>fire engine</td>
<td>dalmatian</td>
<td>gill fungus</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dead-man’s-fingers</td>
<td>grape</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ffordshire bulterrier</td>
<td>elderberry</td>
<td></td>
</tr>
<tr>
<td></td>
<td>currant</td>
<td>howler monkey</td>
<td></td>
</tr>
<tr>
<td></td>
<td>howler monkey</td>
<td>howler monkey</td>
<td></td>
</tr>
</tbody>
</table>
ImageNet Results

[Chart showing classification error over years from 2010 to 2017, with significant improvements in 2015 and 2017. AlexNet is highlighted.]
Recurrent Neural Networks (RNNs)
Recurrent Neural Networks

- Dates back to (Rumelhart *et al.*, 1986)
- A family of neural networks for handling sequential data, which involves variable length inputs or outputs
- Especially, for natural language processing (NLP)
Like Markov model, but here $s^{(t+1)}$ is deterministic given $s^{(t)}$
Computation Graph

\[ s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta) \]
Compact view

\[ s(t+1) = f(s(t), x(t+1); \theta) \]

Key: the same \( f \) and \( \theta \) for all time steps

Like stationary HMM
Recurrent Neural Networks

• Use the same computational function and parameters across different time steps of the sequence

• Each time step: takes the input entry and the previous hidden state to compute the output entry

• Loss: typically computed every time step
Recurrent Neural Networks

Figure from *Deep Learning*, by Goodfellow, Bengio and Courville
Recurrent Neural Networks

There are many variants of RNNs since the functional form to compute $s^{(t)}$ can vary, e.g., LSTM

Math formula:

\[
\begin{align*}
\alpha^{(t)} &= b + W s^{(t-1)} + U x^{(t)} \\
s^{(t)} &= \tanh(\alpha^{(t)}) \\
o^{(t)} &= c + V s^{(t)} \\
y^{(t)} &= \text{softmax}(o^{(t)})
\end{align*}
\]
Sequence-to-Sequence Learning

Example of Neural Machine Translation

Encoding of the source sentence. Provides initial hidden state for Decoder RNN.

Source sentence (input):
les pauvres sont démunis

Target sentence (output):
the poor don't have any money <END>

Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence conditioned on encoding.
Residual Connection

We want deeper and deeper NNs, but going deep is difficult

- Troubles accumulate w/ more layers
- Signals get distorted when propagated
- in forward and backward passes

Commonly used techniques to train “Deep” NNs:

- Weight initialization
- Normalization modules
- Deep residual learning
The Degradation Problem

- Good init + norm enable training deeper models
- Simply stacking more layers?
- Degrade after ~20 layers
- Not overfitting
- Difficult to train

[Graph showing expected accuracy over depth, with a peak at 20 layers and a decline thereafter, labeled "reality." A dashed line from the peak extends to the right, labeled "hope." Polylines connect 10-layer, 20-layer, and hope marks.]
Deep Residual Learning

$$H(x) = F(x) + x$$
Deep Residual Networks (ResNet)

Transformers