CENG 783

Deep Learning

Week – 6
Convolutional Neural Networks
Sinan Kalkan
Motivation

• A fully-connected network has too many parameters
  • On CIFAR-10:
    • Images have size 32x32x3 $\Rightarrow$ one neuron in hidden layer has 3072 weights!
    • With images of size 512x512x3 $\Rightarrow$ one neuron in hidden layer has 786,432 weights!
    • This explodes quickly if you increase the number of neurons & layers.

• Alternative: enforce local connectivity!

Convolution
Formulating Signals in Terms of Impulse Signal

\[ x[0] \delta[n] + x[1] \delta[n-1] + x[-1] \delta[n+1] + x[2] \delta[n-2] \]
Formulating Signals in Terms of Impulse Signal

\[ x[n] = \ldots + x[-2]\delta[n+2] + x[-1]\delta[n+1] + x[0]\delta[n] + x[1]\delta[n-1] + \ldots \]

\[ x[n] = \sum_{k=-\infty}^{+\infty} x[k]\delta[n-k] \]

Important to note the “-” sign

Coefficients \quad Basic Signals

The Sifting Property of the Unit Sample

To sift: Elemek
Unit Sample Response

- Now suppose the system is \textbf{LTI}, and define the \textit{unit sample response} $h[n]$:

$$\delta[n] \rightarrow h[n]$$

\[ \downarrow \]

From Time-Invariance:

$$\delta[n - k] \rightarrow h[n - k]$$

From Linearity:

$$x[n] = \sum_{k=-\infty}^{+\infty} x[k] \delta[n - k] \rightarrow y[n] = \sum_{k=-\infty}^{+\infty} x[k] h[n - k] = x[n] * h[n]$$

convolution sum
Conclusion

The output of any DT LTI System is a convolution of the input signal with the unit-sample response, i.e.

Any DT LTI \[ y[n] = x[n] * h[n] \]

\[ = \sum_{k=-\infty}^{+\infty} x[k] h[n - k] \]

As a result, any DT LTI Systems are completely characterized by its unit sample response.

Power of convolution

- Describe a “system” (or operation) with a very simple function (impulse response).
- Determine the output by convolving the input with the impulse response.
Convolution

• Definition of continuous-time convolution

\[ x(t) * h(t) = \int x(\tau)h(t - \tau) \, d\tau \]

\[ y(t) = x(t) * h(t) = \int_{-\infty}^{+\infty} x(\tau)h(t - \tau) \, d\tau \]

\[ \begin{align*}
  h(\tau) \xrightarrow{\text{Flip}} h(-\tau) \xrightarrow{\text{Slide}} h(t - \tau) \xrightarrow{\text{Multiply}} \\
  x(\tau)h(t - \tau) \xrightarrow{\text{Integrate}} \int_{-\infty}^{+\infty} x(\tau)h(t - \tau) \, d\tau
\end{align*} \]
Convolution

- Definition of discrete-time convolution

\[ x[n] \ast h[n] = \sum x[k]h[n - k] \]

Choose the value of \( n \) and consider it fixed

\[ y[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n - k] \]

View as functions of \( k \) with \( n \) fixed

From \( x[n] \) and \( h[n] \) to \( x[k] \) and \( h[n-k] \)
Discrete-time 2D Convolution

- For images, we need two-dimensional convolution:

\[ s[i, j] = (I * K)[i, j] = \sum_m \sum_n I[m, n]K[i - m, j - n] \]

- These multi-dimensional arrays are called tensors

- We have commutative property:

\[ s[i, j] = (I * K)[i, j] = \sum_m \sum_n I[i - m, j - n]K[m, n] \]

- Instead of subtraction, we can also write (easy to drive by a change of variables). This is called cross-correlation:

\[ s[i, j] = (I * K)[i, j] = \sum_m \sum_n I[i + m, j + n]K[m, n] \]
Example multi-dimensional convolution

Filtering

- Convolutional
  - Dependencies are local
  - Translation equivariance
  - Tied filter weights (few params)
  - Stride 1, 2, … (faster, less mem.)

Input

Feature Map

Slide: R. Fergus
Overview of CNN
CNN layers

- **Stages of CNN:**
  - Convolution (in parallel) to produce pre-synaptic activations
  - Detector: Non-linear function
  - Pooling: A summary of a neighborhood

- **Pooling of a rectangular region:**
  - Max
  - Average
  - L2 norm
  - Weighted average acc. to the distance to the center
  - ...

Regular ANN

CNN

http://cs231n.github.io/convolutional-networks/
An example architecture

http://cs231n.github.io/convolutional-networks/
Convolution in CNN
Convolution in CNN

\[ y(t) = x(t) * h(t) = \int x(\tau)h(t - \tau) \, d\tau \]

- \( x(t) \): input
- \( h(t) \): impulse response (in engineering), kernel (in CNN)
- \( y(t) \): output, feature map (in CNN)
- And \( h(t) = 0 \) for \( t < 0 \). Otherwise, it means it uses future input.

- Effectively, we are learning the filters that best work for our problem.
Convolution in CNN

- The weights correspond to the kernel
- The weights are shared in a channel (depth slice)
- We are effectively learning filters that respond to some part/entities/visual-cues etc.

Local connectivity in CNN = Receptive fields

- Each neuron is connected to only a local neighborhood, i.e., receptive field
- The size of the receptive field ➔ another hyper-parameter.
Having looked at convolution in general and convolution in CNN:

Why convolution in neural networks?

- In regular ANNs, each output node considers all data.
  - Matrix multiplication in essence.
  - This is redundant in most cases.
  - Expensive.

- Instead, a smaller matrix multiplication operation.
  - Sparse interaction.
  - Can extract meaningful entities, such edges, corners, that occupy less space.
  - This reduces space requirements, and improves speed.

- Regular ANN: m x n parameters

- With CNN (if we restrict the output to link to k units): n x k parameters.
When things go deep, an output may depend on all or most of the input:

Having looked at convolution in general and convolution in CNN: Why convolution in neural networks?

- Parameter sharing
  - In regular ANN, each weight is independent

- In CNN, a layer might re-apply the same convolution and therefore, share the parameters of a convolution
  - Reduces storage and learning time

- With ANN: 320x280x320x280 multiplications
- With CNN: 320x280x3 multiplications

Having looked at convolution in general and convolution in CNN: Why convolution in neural networks?

- Equivariant to translation
  - The output will be the same, just translated, since the weights are shared.

- Not equivariant to scale or rotation.
Connectivity in CNN

- **Local**
  - The behavior of a neuron does not change other than being restricted to a subspace of the input.

- Each neuron is connected to slice of the previous layer

- A layer is actually a volume having a certain width x height and depth (or channel)

- A neuron is connected to a subspace of width x height but to all channels (depth)

- Example: CIFAR-10
  - Input: 32 x 32 x 3 (3 for RGB channels)
  - A neuron in the next layer with receptive field size 5x5 has input from a volume of 5x5x3.

http://cs231n.github.io/convolutional-networks/
Important parameters

• Depth (number of channels)
  • We will have more neurons getting input from the same receptive field
  • This is similar to the hidden neurons with connections to the same input
  • These neurons learn to become selective to the presence of different signal in the same receptive field

• Stride
  • The amount of space between neighboring receptive fields
  • If it is small, RFs overlap more
  • If it is big, RFs overlap less

• How to handle the boundaries?
  i. Option 1: Don’t process the boundaries. Only process pixels on which convolution window can be placed fully.
  ii. Option 2: Zero-pad the input so that convolution can be performed at the boundary pixels.

Weights:

http://cs231n.github.io/convolutional-networks/
Padding illustration

- Only convolution layers are shown.
- Top: no padding $\Rightarrow$ layers shrink in size.
- Bottom: zero padding $\Rightarrow$ layers keep their size fixed.

Figure 9.11: The effect of zero padding on network size: Consider a convolutional network with a kernel of width six at every layer. In this example, do not use any pooling, so only the convolution operation itself shrinks the network size. Top) In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not even move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive and some shrinking is inevitable in this kind of architecture. Bottom) By adding five implicit zeroes to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

Size of the network

- Along a dimension:
  - $W$: Size of the input
  - $F$: Size of the receptive field
  - $S$: Stride
  - $P$: Amount of zero-padding

- Then: the number of neurons in the output:

\[
\frac{W - F + 2P}{S} + 1
\]

- If this number is not an integer, your strides are incorrect and your neurons cannot tile nicely to cover the input volume
Size of the network

• Arranging these hyperparameters can be problematic

• Example:

• If $W=10$, $P=0$, and $F=3$, then

$$\frac{W - F + 2P}{S} + 1 = \frac{10 - 3 + 0}{S} + 1 = \frac{7}{S} + 1$$

i.e., $S$ cannot be an integer other than 1 or 7.

• Zero-padding is your friend here.
Real example – AlexNet (Krizhevsky et al., 2012)

• Images: 227x227x3

• W=227, F=11, S=4, P=0  \Rightarrow \frac{227-11}{4} + 1 = 55 \quad (\text{the size of the convolutional layer})

• Convolution layer: 55x55x96 neurons  \quad (96: \text{the depth, the number of channels})

• The neurons in the first layer: 55x55x96 = 290,400 neurons
  • Each has 11x11x3 receptive field  \Rightarrow 363 \text{ weights and 1 bias}
  • Then, 290,400x364 = 105,705,600 parameters just for the first convolution layer (if no sharing)
  • Very high number
Real example – AlexNet (Krizhevsky et al., 2012)

• However, we can share the parameters
  • For each channel (slice of depth), have the same set of weights
  • If 96 channels, this means 96 different set of weights
  • Then, 96x364 = 34,944 parameters
  • 364 weights shared by 55x55 neurons in each channel

Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size $[11 \times 11 \times 3]$, and each one is shared by the $55 \times 55$ neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the $55 \times 55$ distinct locations in the Conv layer output volume.

http://cs231n.github.io/convolutional-networks/