Convolutional and Recurrent Networks

Steve Avsec

Illinois Institute of Technology

March 25, 2024

Overview

1. Convolutional neural networks (CNNs)
2. Recurrent Neural Networks (RNNs)

What is a convolution?

Let $\mathbf{a}, \mathbf{b}$ be two bi-infinite sequences.

What is a convolution?

Let $\mathbf{a}, \mathbf{b}$ be two bi-infinite sequences.

Definition

The *convolution* of $\mathbf{a}$ and $\mathbf{b}$ is given by

$$(\mathbf{a} \star \mathbf{b})(n) = \sum_{m=-\infty}^{\infty} \mathbf{a}(m)\mathbf{b}(n-m)$$

What is a convolution?

Let $\mathbf{a}, \mathbf{b}$ be two bi-infinite sequences.

Definition

The *convolution* of $\mathbf{a}$ and $\mathbf{b}$ is given by

$$(\mathbf{a} \star \mathbf{b})(n) = \sum_{m=-\infty}^{\infty} \mathbf{a}(m)\mathbf{b}(n-m)$$

Why? Typically, this is viewed as an ”averaging” operation between $\mathbf{a}$ and $\mathbf{b}$.

That Sounds Very 1-D

We can lift this up to two dimensions by defining

$$(\mathbf{a} \star \mathbf{b})(m, n) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \mathbf{a}(i, j)\mathbf{b}((m, n) - (i, j))$$

That Sounds Very 1-D

We can lift this up to two dimensions by defining

$$(\mathbf{a} \star \mathbf{b})(m, n) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \mathbf{a}(i, j)\mathbf{b}((m, n) - (i, j))$$

If we keep indexing this way we'll run out of letters so instead

$$(\mathbf{a} \star \mathbf{b})(\mathbf{n}) = \sum_{\mathbf{m} \in \mathbb{Z}^d} \mathbf{a}(\mathbf{m})\mathbf{b}(\mathbf{n} - \mathbf{m})$$

where $d = 1, 2, 3, \ldots$.

Yeah, But Infinity Doesn't Exist

If we have two finite sequences we can

1. Pad zeros out to infinity.

Yeah, But Infinity Doesn't Exist

If we have two finite sequences we can

1. Pad zeros out to infinity.
2. Impose a *Dirichlet* type boundary

$$\mathbf{r}(\mathbf{m}) = c$$

for some constant c and $\mathbf{m}$ on the boundary.

Yeah, But Infinity Doesn't Exist

If we have two finite sequences we can

1. Pad zeros out to infinity.
2. Impose a *Dirichlet* type boundary

$$\mathbf{r}(\mathbf{m}) = c$$

for some constant c and $\mathbf{m}$ on the boundary.

3. Impose a *Neumann* type boundary

$$\partial\mathbf{r}(\mathbf{m}) = c$$

where ∂ is some difference operator and $\mathbf{m}$ is on the boundary and c is some constant.

What does this have to do with Deep Learning?

Convolutional layers lead to three positive attributes in feedforward architectures:

1. Sparse interactions.
   - Example: An image might have a million pixels, but the kernel might only look at a 10 by 10 window

What does this have to do with Deep Learning?

Convolutional layers lead to three positive attributes in feedforward architectures:

1. Sparse interactions.
   - Example: An image might have a million pixels, but the kernel might only look at a 10 by 10 window
2. Parameter sharing.
   - Example: You only have to learn parameters for the 100 parameter kernel in the previous example.

What does this have to do with Deep Learning?

Convolutional layers lead to three positive attributes in feedforward architectures:

1. Sparse interactions.
   - Example: An image might have a million pixels, but the kernel might only look at a 10 by 10 window
2. Parameter sharing.
   - Example: You only have to learn parameters for the 100 parameter kernel in the previous example.
3. Equivariance.
   - If you choose your kernel correctly, images can be rotated, translated, lightened, darkened and the output of the convolution will be the same.

Pooling

Pooling is an operation that takes several ”nearby” inputs and produces an output via some summary statistic.

- Max pooling: Take the maximum of the related inputs.

Pooling

Pooling is an operation that takes several ”nearby” inputs and produces an output via some summary statistic.

- Max pooling: Take the maximum of the related inputs.
- Average pooling

Pooling

Pooling is an operation that takes several ”nearby” inputs and produces an output via some summary statistic.

- Max pooling: Take the maximum of the related inputs.
- Average pooling
- Median pooling

Typical Structure

A ”typical” convolutional layer is composed of

- An affine transformation
  - Example: Difference by one shift to the left.

Typical Structure

A ”typical” convolutional layer is composed of

- An affine transformation
  - Example: Difference by one shift to the left.
- A nonlinear function e.g. ReLU

Typical Structure

A "typical" convolutional layer is composed of

- An affine transformation
  - Example: Difference by one shift to the left.
- A nonlinear function e.g. ReLU
- A pooling operation

Recurrence Relations

Let $\{s_1, \ldots, s_n\}$ be a set of states.

Definition

A recurrence relation is a function of the form

$$s_{t_j} = f(s_{t_j-1}, \theta)$$

where θ is some set of parameters

Recurrence Relations

Let $\{s_1, \ldots, s_n\}$ be a set of states.

Definition

A recurrence relation is a function of the form

$$s_{t_j} = f(s_{t_j-1}, \theta)$$

where θ is some set of parameters

Could instead learn a function like

$$s_{t_j} = g(s_{t_j-1}, \ldots, s_{t_1}, \theta)$$

but such a g can't handle flexible input lengths

Why use these things?

- Language modeling
  - Entity recognition
  - Sentiment analysis
  - Text summarization

Why use these things?

- Language modeling
  - Entity recognition
  - Sentiment analysis
  - Text summarization
- Transaction fraud detection
  - Input: A transaction history for a bunch of shoppers/shops.
  - Training set: A set of transactions labeling fraudulant transactions.
  - Classify most recent transactions given an arbitrary length history.

Why use these things?

- Language modeling
  - Entity recognition
  - Sentiment analysis
  - Text summarization
- Transaction fraud detection
  - Input: A transaction history for a bunch of shoppers/shops.
  - Training set: A set of transactions labeling fraudulant transactions.
  - Classify most recent transactions given an arbitrary length history.
- Time series prediction
  - LSTM (long-short term memory)

Three Design Choices

- Have an output at each time step and have recurrent connections between hidden units.

Three Design Choices

- Have an output at each time step and have recurrent connections between hidden units.
- Have an output at each time step and have recurrent connections only between output of the previous output and next hidden units.

Three Design Choices

- Have an output at each time step and have recurrent connections between hidden units.
- Have an output at each time step and have recurrent connections only between output of the previous output and next hidden units.
- Have connections between hidden units but only produce one output per sequence.

Gradients

We can write down a system:

- $\mathbf{a}_t = \mathbf{b} + W\mathbf{h}_{t-1} + U\mathbf{x}_t$
- $\mathbf{h}_t = g(\mathbf{a})$
- $\mathbf{o}_t = \mathbf{c} + V\mathbf{h}_t$
- $\mathbf{y}_t = d(\mathbf{o}_t)$

Gradients

We can write down a system:

- $\mathbf{a}_t = \mathbf{b} + W\mathbf{h}_{t-1} + U\mathbf{x}_t$
- $\mathbf{h}_t = g(\mathbf{a})$
- $\mathbf{o}_t = \mathbf{c} + V\mathbf{h}_t$
- $\mathbf{y}_t = d(\mathbf{o}_t)$

Here g is some non-linear activation function and d is some decision function (e.g. softmax).

Gradients

We can write down a system:

- $\mathbf{a}_t = \mathbf{b} + W\mathbf{h}_{t-1} + U\mathbf{x}_t$
- $\mathbf{h}_t = g(\mathbf{a})$
- $\mathbf{o}_t = \mathbf{c} + V\mathbf{h}_t$
- $\mathbf{y}_t = d(\mathbf{o}_t)$

Here g is some non-linear activation function and d is some decision function (e.g. softmax).

We can ”unroll” these equations and write down the gradients involved to fit the model.

Bidirectional RNNs

Essentially, train two networks, one forward and one backward and ensemble results.

Bidirectional RNNs

Essentially, train two networks, one forward and one backward and ensemble results.

Why? Language problems, for instance, often have information after the current state that is relevant.

Bidirectional RNNs

Essentially, train two networks, one forward and one backward and ensemble results.

Why? Language problems, for instance, often have information after the current state that is relevant.

State-of-the-art in speech recognition, entity recognition, translation, part-of-speech tagging, etc.

The Problem of Long-Term Behavior

- Take away inputs and outputs and you just have

$$\mathbf{h}_n = W^n \mathbf{h}_0$$

- A matrix W can (almost) always be decomposed by

$$W = Q^{-1} \Lambda Q$$

(eigen-decomposition).

- If all of the eigenvalues have absolute value less than one, W^n tends to the zero matrix.

The Problem of Long-Term Behavior

- Take away inputs and outputs and you just have

$$\mathbf{h}_n = W^n \mathbf{h}_0$$

- A matrix W can (almost) always be decomposed by

$$W = Q^{-1} \Lambda Q$$

(eigen-decomposition).

- If all of the eigenvalues have absolute value less than one, W^n tends to the zero matrix.
- If any of the eigenvalues have absolute value greater than one, W^n explodes in magnitude.