Applied Machine Learning

Convolutional Neural Networks

Siamak Ravanbakhsh

COMP 551 (winter 2020)
Learning objectives

understand the convolution layer and the architecture of conv-net

- its inductive bias
- its derivation from fully connected layer
- different types of convolution
**MLP and image data**

we can apply an MLP to image data

first vectorize the input  \( x \rightarrow \text{vec}(x) \in \mathbb{R}^{784} \)

feed it to the MLP (with L layers) and predict the labels

\[
\text{softmax} \circ W^L \circ \ldots \circ \text{ReLU} \circ W^1 \text{vec}(x)
\]

the model knows nothing about the image structure

we could **shuffle all pixels** and learn an MLP with similar performance

how to bias the model, so that it "knows" its input is image?

image is like 2D version of sequence data

lets find the right model for sequence first...
suppose we want to convert one sequence to another \( \mathbb{R}^D \to \mathbb{R}^D \)

suppose we have a dataset of input-output pairs \( \{(x^{(n)}, y^{(n)})\}_n \)

consider only a single layer \( y = g(Wx) \)

we may assume, each output unit is the same function shifted along the sequence

when is this a good assumption?

elements of \( w \) of the same color are tied together

(parameter-sharing)
we may assume, each output unit is the same function shifted along the sequence

we may further assume each output is a local function of input

larger receptive field with multiple layers

size of the receptive field is 3

size of the receptive field is 5
Cross-correlation (1D)

we may further assume each output is a local function of input

\[ w = [w_1, \ldots, w_K] = [W_{c,c-\lfloor \frac{K}{2} \rfloor}, \ldots, W_{c,c+\lfloor \frac{K}{2} \rfloor}] \]

instead of the whole matrix we can keep the one set of nonzero values

we can write matrix multiplication as cross-correlation of \( w \) and \( x \)

\[ y_c = g\left( \sum_{d=1}^{D} W_{c,d} x_d \right) = g\left( \sum_{k=1}^{K} w_k x_{c-\lfloor \frac{K}{2} \rfloor} + k \right) \]

slide on the input, calculate inner product and apply the nonlinearity
**Convolution (1D)**

Cross-correlation is similar to convolution

\[
y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c + k)
\]

Cross-correlation

\[
y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c + k)
\]

Convolution

\[
y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c - k)
\]

Convolution flips \(w\) or \(x\) (to be commutative)

\[
y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c - k)
\]

Change of variable

\[
y(c) = \sum_{d=-\infty}^{\infty} w(c - d)x(d)
\]

since we learn \(w\), flipping it makes no difference

in practice, we use cross correlation rather than convolution

convolution is equivariant wrt translation

-- *i.e.*, shifting \(x\), shifts \(w*x\)
Convolution (2D)

similar idea of parameter-sharing and locality extends to 2 dimension (i.e. image data)

\[ y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{d_1+k_1-1,d_2+k_2-1} w_{k_1,k_2} \]
Convolution (2D)

similar idea of parameter-sharing and locality extends to 2 dimension (i.e. image data)

\[ y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{d_1+k_1-1,d_2+k_2-1} w_{k_1,k_2} \]

there are different ways of handling the borders

- zero-pad the input, and produce all non-zero outputs (full)
- output is larger than input (by how much?)
- each input participates in the same number of output elements

output length (for one dimension)
\[ |D + \text{padding} - K + 1| \]

zero-pad the input, to keep the output dims similar to input (same)

no padding at all (valid)
- output is small than input (how much?)

image credit: Vincent Dumoulin, Francesco Visin
Pooling

sometimes we would like to reduce the size of output e.g., from $D \times D$ to $D/2 \times D/2$
a combination of pooling and downsampling is used

1. calculate the output $\tilde{y}_d = g\left(\sum_{k=1}^{K} x_{d+k-1} w_k\right)$
2. aggregate the output over different regions

\[
y_d = \text{pool}\{\tilde{y}_d, \ldots, \tilde{y}_{d+p}\}
\]

*two common aggregation functions are max and mean*

pooling results in some degree of invariance to translation

3. often this is followed by subsampling using the same step size

the same idea extends to higher dimensions
Strided convolution

Alternatively we can directly subsample the output

\[
\tilde{y}_d = g \left( \sum_{k=1}^{K} x_{(d-1)+k} w_k \right)
\]

\[
y_d = \tilde{y}_{dp}
\]

equivalent to
Strided convolution

the same idea extends to higher dimensions

\[ y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{p_1(d_1-1)+k_1, p_2(d_2-1)+k_2} w_{k_1,k_2} \]

with padding

Different step-sizes for different dimensions

output length (for one dimension)

\[ \left\lfloor \frac{D + \text{padding} - K}{\text{stride}} \right\rfloor + 1 \]

image: Dumoulin & Visin’16
Channels

so far we assumed a single input and output sequence or image

with RGB data, we have 3 input channels ( \( M = 3 \) )
this example: 2 input channels

\[
x \in \mathbb{R}^{M \times D_1 \times D_2}
\]

we have one \( K_1 \times K_2 \) filters per input-output channel combination

\[
w \in \mathbb{R}^{M \times M' \times K_1 \times K_2}
\]

\[
+ \text{ add the result of convolution from different input channels}
\]

similarly we can produce multiple output channels \( M' = 3 \)

\[
y \in \mathbb{R}^{M' \times D'_1 \times D'_2}
\]

image: Dumoulin & Visin'16
Channels

so far we assumed a single input and output sequence or image

we can also add a bias parameter (b), one per each output channel

\[ b \in \mathbb{R}^{M'} \]

\[
y_{m',d_1,d_2} = g\left( \sum_{m=1}^{M} \sum_{k_1} \sum_{k_2} w_{m,m',k_1,k_2} x_{m,d_1+k_1-1,d_2+k_2-1} + b_{m'} \right)
\]

\[
y \in \mathbb{R}^{M' \times D'_1 \times D'_2}
\]

\[
x \in \mathbb{R}^{M \times D_1 \times D_2}
\]

\[
w \in \mathbb{R}^{M \times M' \times K_1 \times K_2}
\]

image: https://cs231n.github.io/convolutional-networks/
**Convolutional Neural Network (CNN)**

CNN or convnet is a neural network with convolutional layers (so it's a special type of MLP)
it could be applied to 1D sequence, 2D image or 3D volumetric data
**example:** conv-net architecture (derived from AlexNet) for image classification

- **Input**
- **Conv1**
- **Conv2**
- **Conv3**
- **Conv4**
- **Conv5**
- **Fully connected layers**: FC6, FC7, FC8
- **Number of classes**: 1000

Visualization of the convolution kernel at the first layer: 11x11x3x96
96 filters, each one is 11x11x3. each of these is responsible for one of 96 feature maps in the second layer
CNN or convnet is a neural network with convolutional layers (so it's a special type of MLP) it could be applied to 1D sequence, 2D image or 3D volumetric data

**example:** conv-net architecture (derived from AlexNet) for image classification

Deeper units represent more abstract features.
Application: image classification

Convnets have achieved super-human performance in image classification

ImageNet challenge: >1M images, 1000 classes

Application: image classification

variety of increasingly deeper architectures have been proposed

Application: image classification

variety of increasingly deeper architectures have been proposed
**Training:** backpropagation through convolution

consider the strided 1D convolution op.  \( y_{m',d} = \sum_m \sum_k w_{m,m',k} x_{m,p(d-1)+k} \)

using backprop. we have \( \frac{\partial J}{\partial y_{m',d'}} \) so far and we need

1) \( \frac{\partial y_{m',d'}}{\partial w_{m,m',k}} \) so as to get the gradients

\[
\frac{\partial J}{\partial w_{m,m',k}} = \sum_{d'} \frac{\partial J}{\partial y_{m',d'}} \frac{\partial y_{m',d'}}{\partial w_{m,m',k}}
\]

2) \( \frac{\partial y_{m',d'}}{\partial x_{m,d}} \) to backpropagate to previous layer

\[
\frac{\partial J}{\partial x_{m,d}} = \sum_{d',m'} \frac{\partial J}{\partial y_{m',d'}} \frac{\partial y_{m',d'}}{\partial x_{m,d}}
\]

this operation is similar to multiplication by transpose of the parameter-sharing matrix (transposed convolution)
Naive implementation

consider the strided 1D convolution op. with stride 1. and single input-output channels

\[ y_d = \sum_k w_k x_{d+k-1} \]

in practice most efficient implementation depends on the filter size (using FFT for large filters)

**forward pass**

```python
1 def Conv1D( 2 x, # D (length) 3 w, # K (filter length) 4 ); 5 6 D, = x.shape 7 K, = w.shape 8 Dp = D - K + 1 # output length 9 y = np.zeros((Dp)) 10 for dp in range(Dp): 11 y[dp] = np.sum(x[dp:dp+K] * w) 12 return y
```

**backward pass**

```python
1 def Conv1DBackProp( 2 x, # D (length) 3 w, # K 4 dJdy,#Dp: error from layer above 5 ); 6 7 D, = x.shape 8 K, = w.shape 9 Dp = dJdy.shape 10 dw = np.zeros_like(w) 11 dJdx = np.zeros_like(x) 12 for dp in range(Dp): 13 dw += np.sum(dJdy[dp] * x[dp:dp+K], 14 dJdx[dp:dp+K] += dJdy[dp:dp+K] * w 15 return dJdx, dw # error to layer below and weight update
```
Transposed Convolution

Transposed convolution (aka deconvolution) recovers the shape of the original input

Convolution with **no stride** and its transpose

no padding of the original convolution corresponds to full padding of in transposed version

Convolution with **stride** and its transpose

this can be used for up-sampling (opposite of stride/pooling) as expected the transpose of a transposed convolution is the original convolution

image: Dumoulin & Visin’16
Dilated Convolution

Dilated (aka atrous) convolution

this can be used to create exponentially large receptive field in few layers

dilation = 8, size of receptive field = 31

dilation = 4, size of receptive field = 15

dilation = 2, size of receptive field = 7

dilation = 1 (i.e., no dilation), size of receptive field = 3

in contrast to stride, dilation does not lose resolution

output length (for one dimension)

\[
\left\lfloor \frac{D + \text{padding} - \text{dilation} \times (K-1) - 1}{\text{stride}} + 1 \right\rfloor
\]

1 torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')

image credits: Kalchbrenner et al'17, Dumoulin & Visin'16
Structured Prediction

the output itself may have (image) structure (e.g., predicting text, audio, image)

example in (semantic) segmentation, we classify each pixel

loss is the sum of cross-entropy loss across the whole image

variety of architectures... one that performs well is **U-Net**

transposed convolution (upconv), concatenation, and skip connection are common in architecture design

architecture search (i.e., combinatorial hyper-parameter search) is an expensive process and an active research area

image: https://sthalles.github.io/deep_segmentation_network/
Summary

convolution layer introduces an **inductive bias** to MLP 
**equivariance** as an inductive bias:
  - translation of the same model is applied to produce different outputs (pixels)
  - the layer is equivariant to **translation**
  - achieved through **parameter-sharing**

**conv-nets** use combinations of
  - convolution layers
  - ReLU (or similar) activations
  - pooling and/or stride for down-sampling
  - skip-connection and/or batch-norm to help with optimization / regularization
  - potentially fully connected layers in the end

**training**
  - backpropagation (similar to MLP)
  - SGD or its improved variations with adaptive learning rate
  - monitor the validation error for early stopping