Neural Network

UW CSE 473
Feb/23/2022
Beibin Li
Summary

- NN
- CNN
  - Image Classification
  - Semantic Segmentation
- RNN
- Transformer
- RL
- PyTorch
Introduction

Accuracy improves

Speed reduces

Energy consumption increases

ImageNet Classification

Top-5 Accuracy (%)

8 Layers

22 Layers

151 Layers

Network Depth
CNN

convolution + nonlinearity

max pooling

convolution + pooling layers

vec

fully connected layers

Nx binary classification

bird

P_{bird}

sunset

P_{sunset}

dog

P_{dog}

cat

P_{cat}
Convolution Operation

Nowadays, we learn kernels from the data.
Learning

\[
\begin{align*}
O_{11} &= F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22} \\
O_{12} &= F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23} \\
O_{21} &= F_{11}X_{21} + F_{12}X_{22} + F_{21}X_{31} + F_{22}X_{32} \\
O_{22} &= F_{11}X_{22} + F_{12}X_{23} + F_{21}X_{32} + F_{22}X_{33}
\end{align*}
\]

Details:
- [https://www.slideshare.net/EdwinEfranJimnezLepe/example-feedforward-backpropagation](https://www.slideshare.net/EdwinEfranJimnezLepe/example-feedforward-backpropagation)
Pooling

e.g. kernel size = 2, stride = 2 for both width and height.
CNN Structures
Image Classification
Image Classification

28 x 28 = [28]^2


CU -> DU -> CU -> DU -> CU -> GAP -> FC -> Cat

CU: Convolutional Unit
DU: Down-sampling Unit
FC: Fully-connected Or Linear Layer
GAP: Global Avg. Pooling

28 x 28
Convolutional Unit (CU) - VGG

Basic Block in ResNet


- Residual Connection
- Element-wise addition of input and output
- Improves gradient flow and accuracy
- In ResNet-18 and ResNet-34
- Still computationally expensive
  - Hard to train very deep networks (> 100 layers)
Bottleneck in ResNet

- Used in ResNet-50, ResNet-101, ResNet-152, etc...
- Computationally Efficient

Influence:
- Bottleneck unit with Depth-wise convs
  - MobileNetv2
  - ShuffleNetv2

CNN Structures
Semantic Segmentation
Encoder-Decoder

Input

Encoding

Encoded Representation

Decoding

Output
Encoder-Decoder in Semantic Segmentation

- **CU**: Convolutional Unit
- **UU**: Up-sampling Unit
- **DU**: Down-sampling Unit
- **FC**: Fully-connected or Linear Layer
- **GAP**: Global Avg. Pooling
U-Net

- Convolutional Unit (CU)
- Down-sampling Unit (DU)
- Fully-connected or Linear Layer (FC)
- Up-sampling Unit (UU)
- Global Avg. Pooling (GAP)
CNN Structures
Detection
RNN Structures
Challenges for time-series signals

- Different signal length
- Online inference for new timepoint

(Vanilla) Neural Network
1940s - 1980s

Hidden Markov Model
Andrew Viterbi, 1967
Lawrence Rabiner, 1989

\( X_t : \) hidden state variables
\( Y_{ti} : i^{th} \) observed variable @ t
Graded state machines
Servan-Schreiber, Cleeremans, and McClelland (1991)

Learning internal representations by error propagation
Rumelhart, Hinton, and Williams (1985)

A parallel distributed processing approach
Jordan (1986)
Recurrent Neural Network
Parameters in Recurrent Neural Network

- $x_t$: input vector ($m \times 1$).
- $h_t$: hidden layer vector ($n \times 1$).
- $o_t$: output vector ($n \times 1$).
- $b_h$: bias vector ($n \times 1$).
- $U, W$: parameter matrices ($n \times m$).
- $V$: parameter matrix ($n \times n$).
- $\sigma_h, \sigma_y$: activation functions.

$h_t = \sigma_h(i_t) = \sigma_h(U_h x_t + V_h h_{t-1} + b_h)$

$y_t = \sigma_y(a_t) = \sigma_y(W_y h_t + b_h)$
LSTM and GRU: Memory for RNNs

LSTM (Long-Short Term Memory)

GRU (Gated Recurrent Unit)

http://dprogrammer.org/rnn-lstm-gru
https://towardsdatascience.com/grus-and-lstm-s-741709a9b9b1
Seq-2-Seq

- Encoder maps a variable-length source sequence (input) to a fixed-length vector
- Decoder maps the vector representation back to a variable-length target sequence (output)
- Two RNNs are trained jointly to maximize the conditional probability of the target sequence given a source sequence
Seq-2-Seq with Attention
Transformer
Limitations of CNN and RNN

1. “Locality” of the convolution operation
   a. Reduce dimension (compared to fully-connected layers) while maintaining useful local information
   b. It could NOT see two pixels that are far away

1. “Recurrentness” of recurrent neural network
   a. It can take an input with arbitrary size (length)
   b. “Vanishing of gradient” problem when sequence length is too long (during backpropagation)
Well, forget about convolution and recurrent

**IS ALL YOU NEED**
What is attention?

Psychology

Eye-tracking

Computer Vision (Saliency Map)

Computer Vision (Backpropagation)

Neural Network (weights)

NLP (align)  
Bahdanau, Cho, Bengio, 2015, ICLR  

Multi-Head Attention

Value, Key, Query  

Transformer
Attention$(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
Vision Transformer

Class
Bird
Ball
Car...

MLP Head

Transformer Encoder

Patch + Position Embedding
* Extra learnable [class] embedding

Linear Projection of Flattened Patches
Limitations of Transformer

1. It cannot learn hierarchical features efficiently (while CNN can)
2. It cannot model periodic finite-state language (while RNN can)
3. It requires lots for computer memory
4. It requires more training data than CNN/RNN (not a big problem)
NN for RL
Which Direction

Regression:

\[
\text{Angle} = [-540°, 540°]
\]

Classification:

- Turn left
- Turn right
- Stay Still
Which Move

Convolution

Fully connected

L0 (Input) 512x512
L1 256x256
L2 128x128
L3 64x64
L4 32x32
F5
F6 (Output)

Go example creation:
Bob van den Hoek

- border fight
- attack
- center ko
- nobi
- hane
- split shape
Design Utility Function

We can train a CNN to replace the manual utility function

Utility = CNN( )
**AlphaZero**

ResNet backbone
Policy Network
Value Network
Monte Carlo Tree Search

Active Learning to balance Exploration v.s. Exploitation
Intro to PyTorch
Deep Learning Frameworks

- Before 2012: custom C++, MatLab, R, Lua, ... code.
  - Only limited libraries/functions
  - You need to do most things yourself
- MXNet (2015)
- TensorFlow (2015)
- Caffe (2015)
- Torch (2002): Lua
- PyTorch (2016)
Why PyTorch

- Autograd
- Dynamic computational graph
- Debugging is easier!
- Data Parallelism (multiple GPU)
- Pythonic-syntax (Python)
- Multiple language support: Python, C++, Java
- Many more!
Machine Learning Process

Define the Problem → Collect Data → Train Model → Evaluate → Deploy

Breast Cancer
Model Definition

```python
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
```
Training a model

```python
running_loss = 0.0
for i, data in enumerate(trainloader, 0):
    # get the inputs
    inputs, labels = data

    # zero the parameter gradients
    optimizer.zero_grad()

    # forward + backward + optimize
    outputs = net(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

    # print statistics
    running_loss += loss.item()
    if i % 2000 == 1999:  # print every 2000 mini-batches
        print('[%d, %5d] loss: %.3f' %
              (epoch + 1, i + 1, running_loss / 2000))
    running_loss = 0.0
```