循环神经网络

彭小江 副教授
深圳技术大学 大数据与互联网学院

邮箱: pengxiaojiang@sztu.edu.cn
个人主页: pengxj.github.io

前馈网络的一些不足

- 连接存在层与层之间，每层的节点之间（自己与自己）是无连接的。（无循环）
- 输入和输出的维数都是固定的，不能任意改变。无法处理变长的序列数据
- 假设每次输入都是独立的，也就是说每次网络的输出只依赖于当前的输入

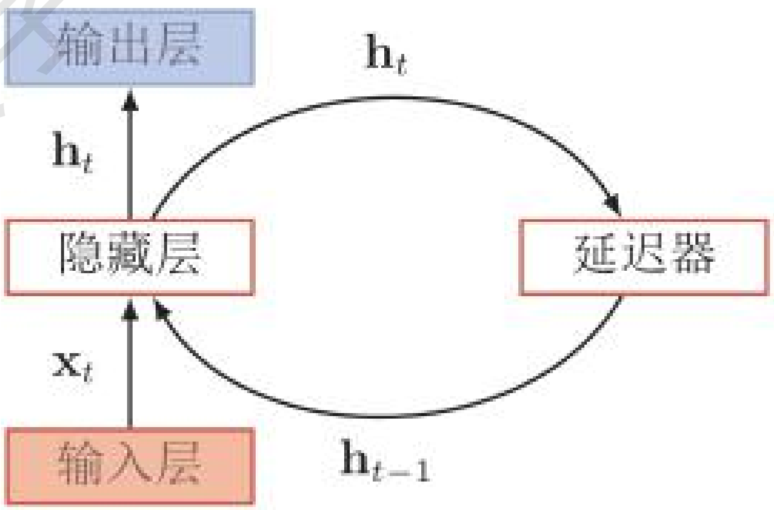
循环神经网络

- 循环神经网络通过使用带自反馈的神经元，能够处理任意长度的序列
- 循环神经网络比前馈神经网络更加符合生物神经网络的结构

循环神经网络

给定一个输入序列 $\mathbf{x}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$，循环神经网络通过下面公式更新带反馈边的隐藏层的**活性值** $\mathbf{h}_t$：

$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otherwise} \end{cases}$$



$$\forall t = 1, \ldots, T(x),\ h_t = \Phi(x_t, h_{t-1}; w),$$

$$\Phi(\cdot\,; w) : \mathbb{R}^D \times \mathbb{R}^Q \to \mathbb{R}^Q.$$

如： $h_t = \mathrm{ReLU}\left(W_{(x\ h)} x_t + W_{(h\ h)} h_{t-1} + b_{(h)}\right)$

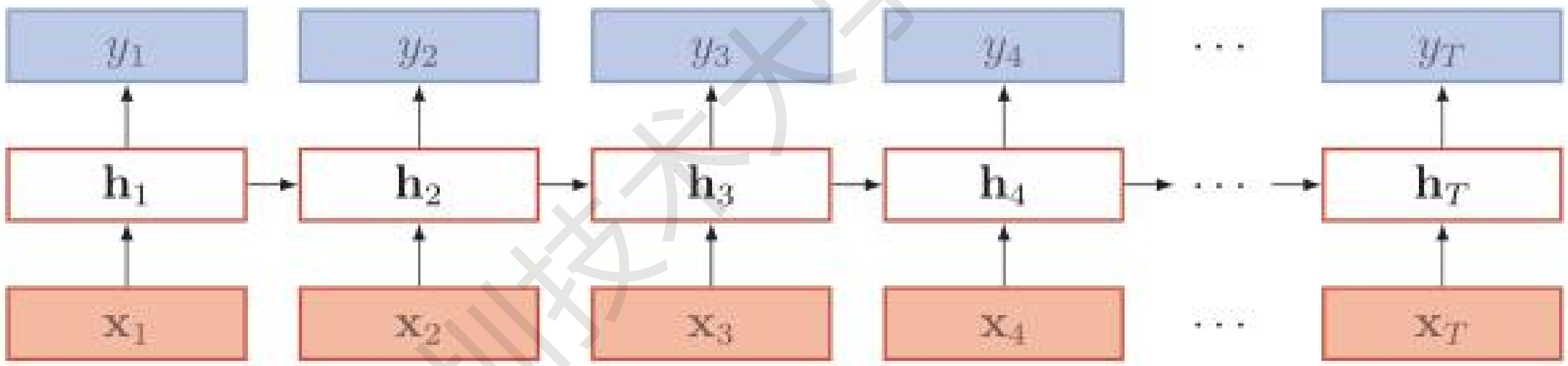
简单循环网络

- 网络输出：

$$y_t = \Psi(h_t; w)$$

如： $y_T = W_{(h\ y)} h_T + b_{(y)}.$

$$\Psi(\cdot; w) : \mathbb{R}^Q \to \mathbb{R}^C.$$

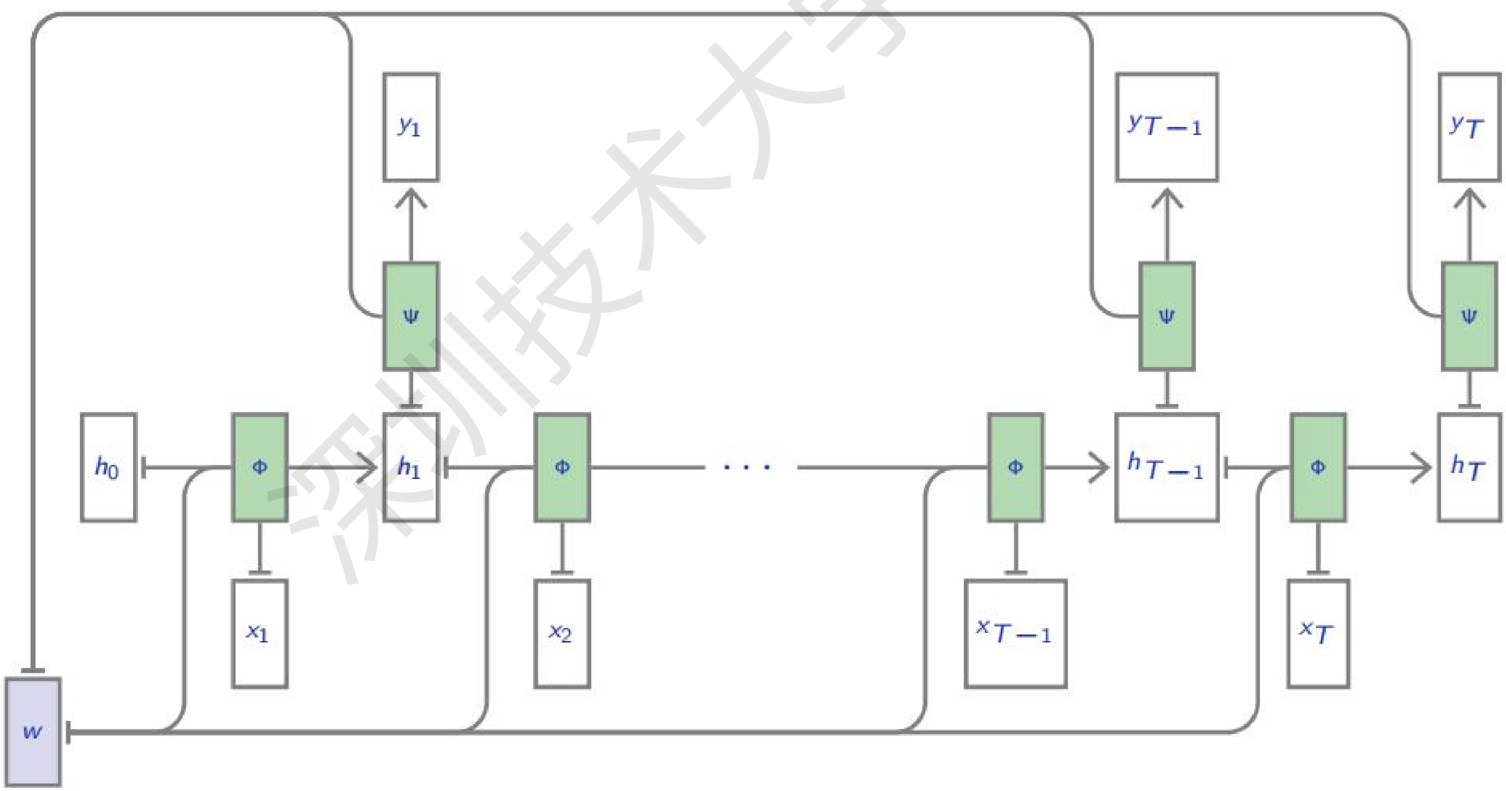


RNN的时序展开形式

前向神经网络：模拟任何函数
RNN：模拟任何程序（计算过程）

RNN的计算图

- 可通过类似的梯度下降进行优化：
  - “backpropagation through time BPTT” (Werbos, 1988).



RNN单元从零开始实现

- 定义所需权重进行计算

从零开始实现

```python
import torch
import torch.nn.functional as F
X, W_xh = torch.randn(1, 5), torch.randn(5, 4) #X为5维数据，时序为1
H, W_hh = torch.randn(1, 4), torch.randn(4, 4)
b = torch.ones(1,1)
h_t = F.relu(torch.matmul(X, W_xh) + torch.matmul(H, W_hh) + b)
h_t
```

$$h_t = \mathrm{ReLU}\left(W_{(x\ h)} x_t + W_{(h\ h)} h_{t-1} + b_{(h)}\right)$$

$$y_T = W_{(h\ y)} h_T + b_{(y)}.$$

RNN用Pytorch模块实现

- 矩阵乘法在pytorch内可以用线性层表示

Pytorch模块定义

```python
import torch.nn as nn
class RnnNet(nn.Module):
    def __init__(self, dim_input, dim_hidden, dim_output):
        super(RnnNet, self).__init__()
        self.fc_x2h = nn.Linear(dim_input, dim_hidden)
        self.fc_h2h = nn.Linear(dim_hidden, dim_hidden, bias = False)
        self.fc_h2y = nn.Linear(dim_hidden, dim_output)  #4x1
        self.dim_hidden = dim_hidden

    def forward(self, x):
        h = x.new_zeros(1, self.dim_hidden)
        for t in range(x.size(0)):
            h = F.relu(self.fc_x2h(x[t:t+1]) + self.fc_h2h(h)) #
        return self.fc_h2y(h)

rnn = RnnNet(5, 4, 4)
t = torch.randn(20, 5) #时序长为20
rnn(t)
```

$$h_t = \mathrm{ReLU}\left(W_{(x\ h)} x_t + W_{(h\ h)} h_{t-1} + b_{(h)}\right)$$

$$y_T = W_{(h\ y)} h_T + b_{(y)}.$$

Pytorch自带的RNN模块

- torch.nn.RNN　　　　RNNCell

```
rnn = nn.RNN(10, 20, 2, batch_first=True) # inputsize, hidden size, num_layers
input = torch.randn(3, 5, 10) # batchsize 3  时序长度为5, 10: 特征维度
h0 = torch.randn(2, 3, 20) # 层数, batchsize, hiddensize
output, hn = rnn(input, h0)  #
print(hn.shape)
output.shape  # W_hy x H  输出size默认为h的维度
```

使用RNNCell进行单个样本运算

```
rnn = nn.RNNCell(10, 20)
input = torch.randn(6, 3, 10)
hx = torch.randn(3, 20)
output = []
for i in range(input.size(0)):
    hx = rnn(input[i], hx)
    output.append(hx)
output[0].shape
```

长期依赖问题

- 循环神经网络在时间维度上非常深！
  - 梯度消失或梯度爆炸

- 如何改进？
  - 梯度爆炸问题
    - 权重衰减（weight decay）
    - 梯度截断
  - 梯度消失问题
    - 改进模型

长期依赖问题

- 改进方法

$$h_t = \text{ReLU}\left(W_{(x\ h)}x_t + W_{(h\ h)}h_{t-1} + b_{(h)}\right)$$

  - 循环边改为线性依赖关系

$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t; \theta),$$

时序上加短路，类似残差网络

  - 增加非线性

$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta),$$

Gating/门控技术

- 当前状态：前一状态和一个中间更新值进行加权平均
- 权重z（f函数）依赖于当前输入和前一状态，该权重也称遗忘门（forget gate)

$$\bar{h}_t = \Phi(x_t, h_{t-1})$$
$$z_t = f(x_t, h_{t-1})$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \bar{h}_t$$

Gating/门控技术

- 当前状态：前一状态和一个中间更新值（full update）进行加权平均
- 权重z（f函数）依赖于当前输入和前一状态，该权重也称遗忘门（forget gate）

$$\bar{h}_t = \Phi(x_t, h_{t-1})$$
$$z_t = f(x_t, h_{t-1})$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \bar{h}_t$$

- 例如：

$$\bar{h}_t = \mathrm{ReLU}\left(W_{(x\ h)} x_t + W_{(h\ h)} h_{t-1} + b_{(h)}\right) \qquad \text{(full update)}$$
$$z_t = \mathrm{sigm}\left(W_{(x\ z)} x_t + W_{(h\ z)} h_{t-1} + b_{(z)}\right) \qquad \text{(forget gate)}$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \bar{h}_t \qquad \text{(recurrent state)}$$

Gating/门控技术

- 当前状态：前一状态和一个中间更新值进行加权平均
- 权重z（f函数）依赖于当前输入和前一状态，该权重也称遗忘门（forget gate）

gating RNN

```
class RecNetWithGating(nn.Module):
    def __init__(self, dim_input, dim_recurrent, dim_output):
        super(RecNetWithGating, self).__init__()
        self.fc_x2h = nn.Linear(dim_input, dim_recurrent)
        self.fc_h2h = nn.Linear(dim_recurrent, dim_recurrent, bias = False)
        self.fc_x2z = nn.Linear(dim_input, dim_recurrent)
        self.fc_h2z = nn.Linear(dim_recurrent, dim_recurrent, bias = False)
        self.fc_h2y = nn.Linear(dim_recurrent, dim_output)
        self.dim_hidden = dim_recurrent
    def forward(self, input):
        h = input.new_zeros(1, self.dim_hidden)
        for t in range(input.size(0)):
            z = torch.sigmoid(self.fc_x2z(input[t:t+1]) + self.fc_h2z(h))
            hb = F.relu(self.fc_x2h(input[t:t+1]) + self.fc_h2h(h))
            h = z * h + (1 - z) * hb
        return self.fc_h2y(h)

rnn = RecNetWithGating(5, 4, 4)
t = torch.randn(20, 5) #时序长为20
rnn(t)
```

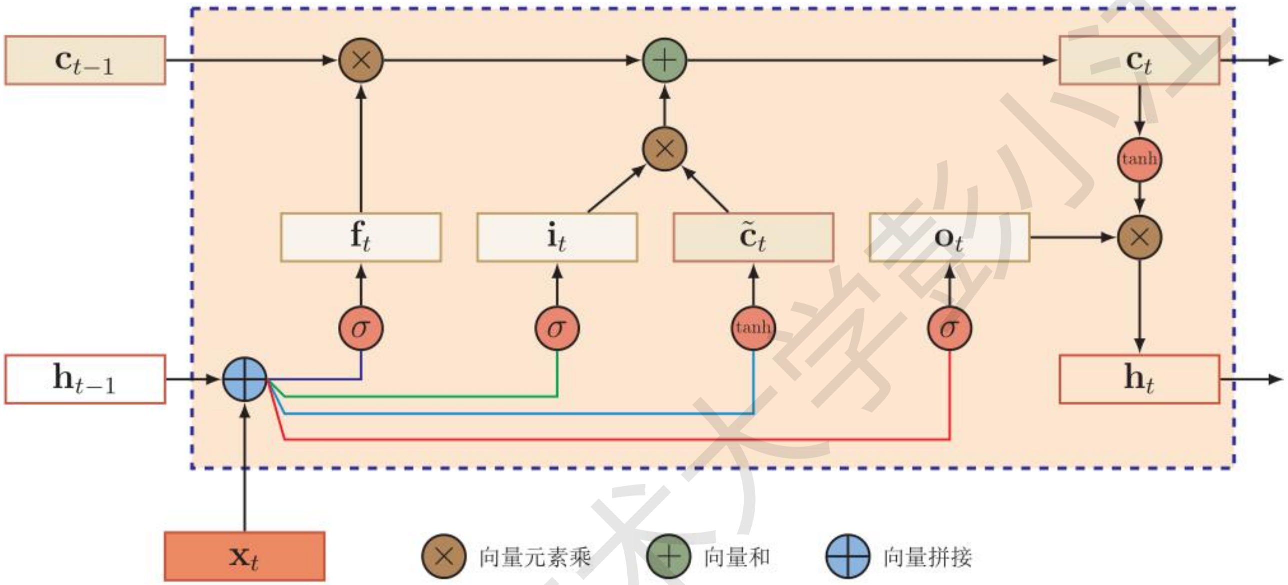
长短时记忆神经网络：LSTM

- Jürgen Schmidhuber （1997）发明的LSTM单元为以下形态：

$$c_t = c_{t-1} + i_t \odot g_t$$

- c为隐含状态，g_t为中间更新值，i_t为门控函数
- 后续Gers等人（2000）和Jozefowicz等人(2015)对其进行了改进，引入了cell状态c，输出状态h，输入门i，输出门o，遗忘门f

长短时记忆单元：LSTM unit



$$f_t = \text{sigm}\left(W_{(x\ f)}x_t + W_{(h\ f)}h_{t-1} + b_{(f)}\right) \qquad \text{(forget gate)}$$

$$i_t = \text{sigm}\left(W_{(x\ i)}x_t + W_{(h\ i)}h_{t-1} + b_{(i)}\right) \qquad \text{(input gate)}$$

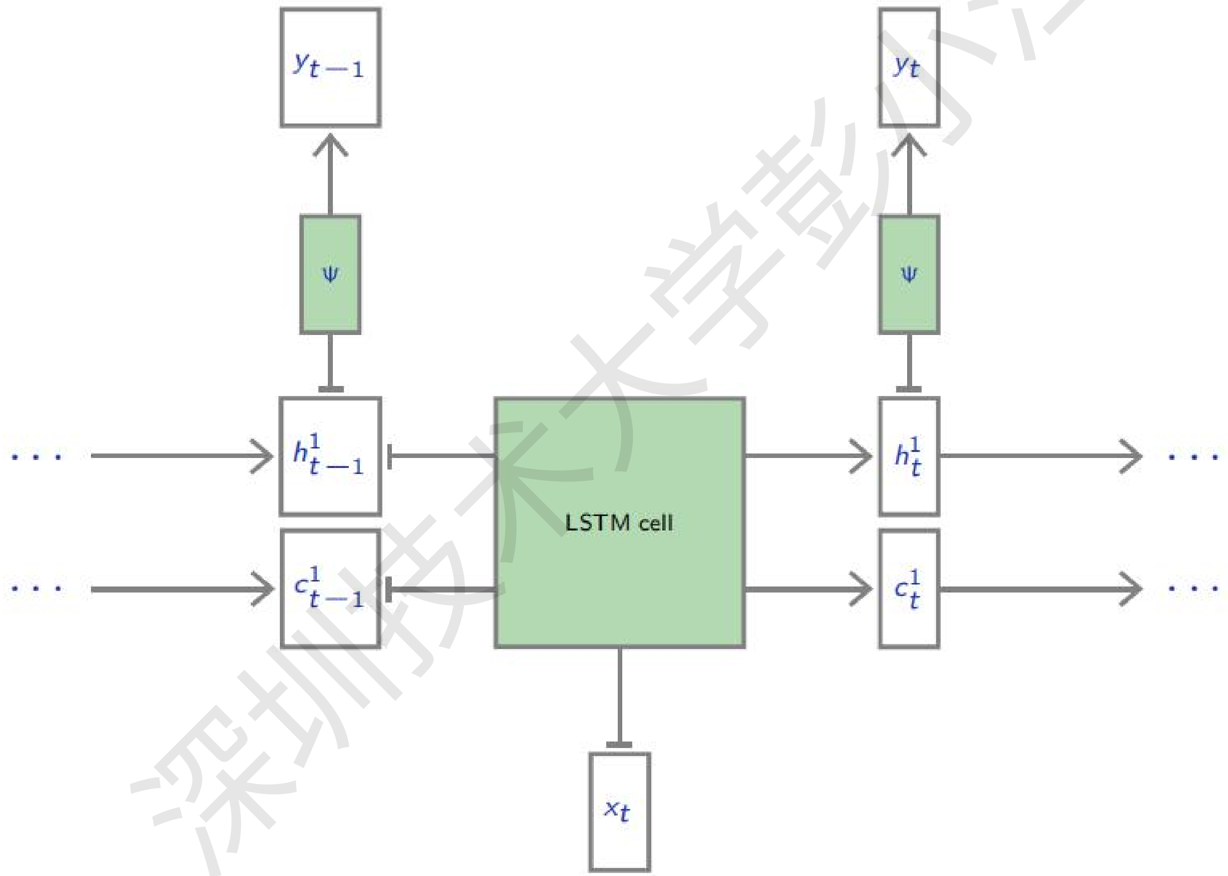
$$g_t = \tanh\left(W_{(x\ c)}x_t + W_{(h\ c)}h_{t-1} + b_{(c)}\right) \qquad \text{(full cell state update)}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \qquad \text{(cell state)}$$

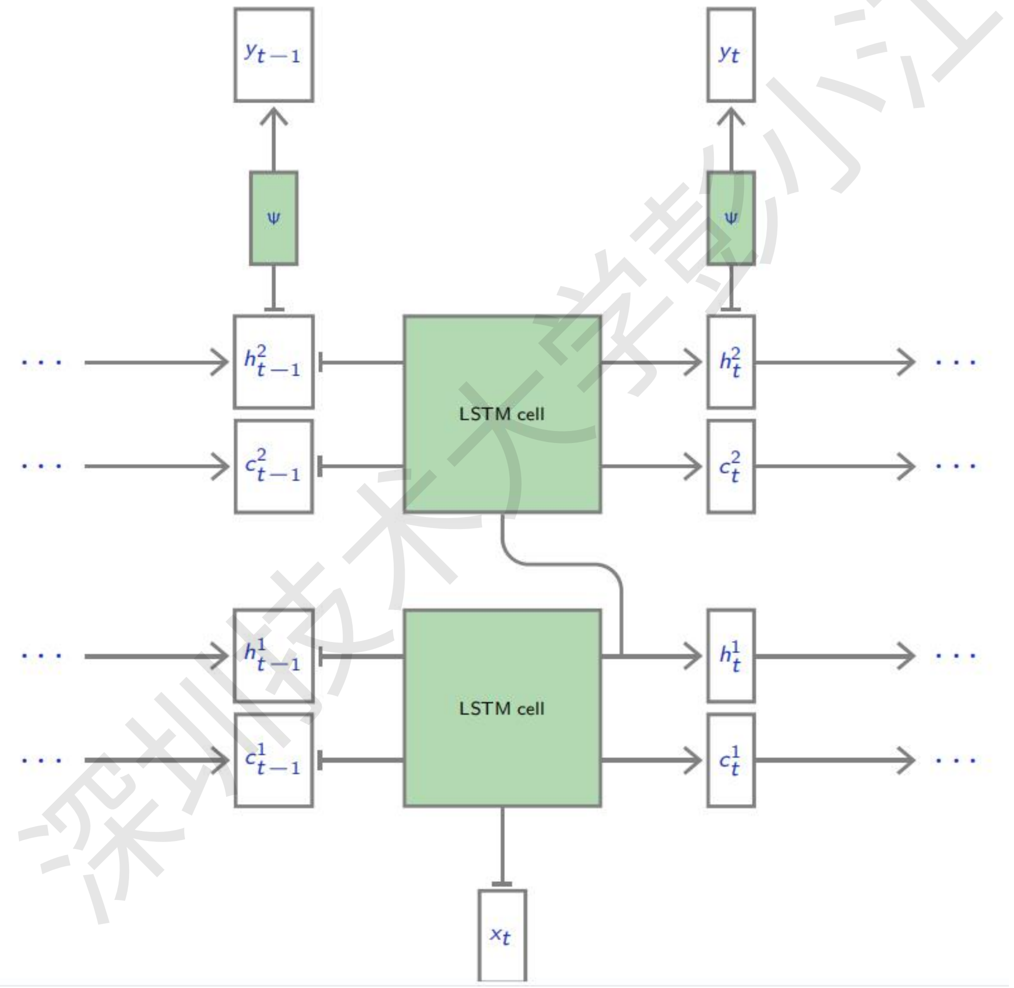
$$o_t = \text{sigm}\left(W_{(x\ o)}x_t + W_{(h\ o)}h_{t-1} + b_{(o)}\right) \qquad \text{(output gate)}$$

$$h_t = o_t \odot \tanh(c_t) \qquad \text{(output state)}$$

长短时记忆网络



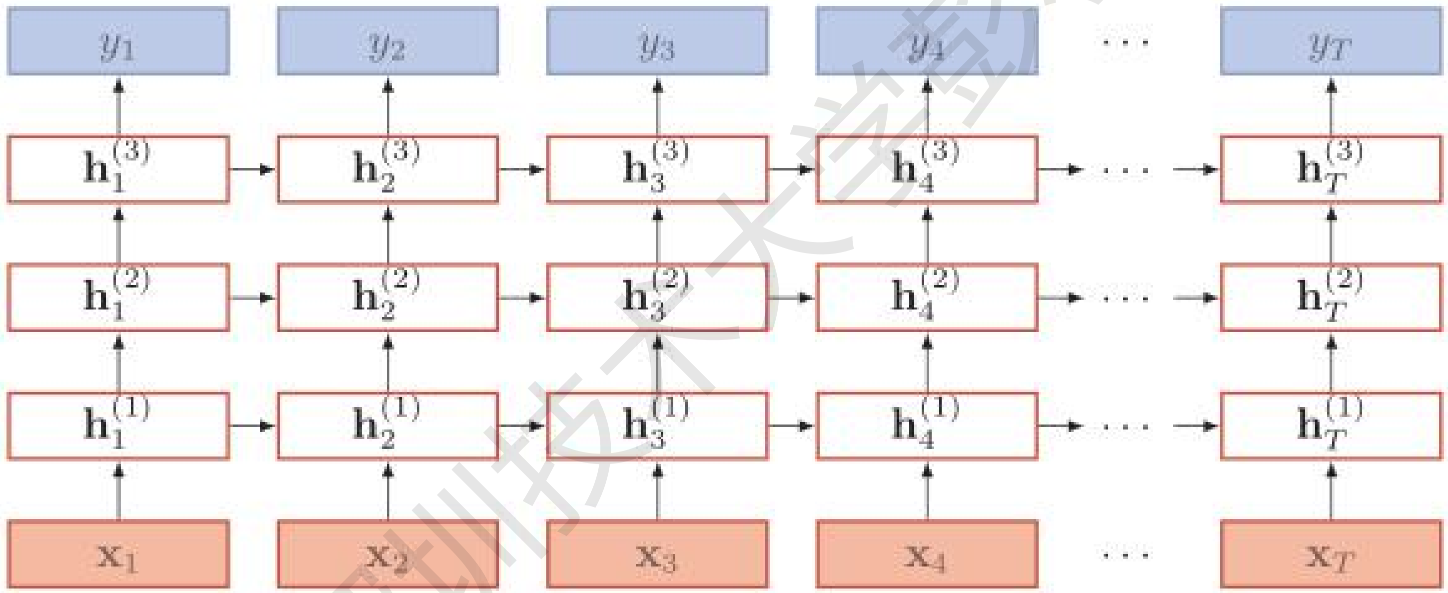
长短时记忆网络（多层）



y_{t-1}
y_t
ψ
ψ
h^2_{t-1}
c^2_{t-1}
LSTM cell
h^2_t
c^2_t
h^1_{t-1}
c^1_{t-1}
LSTM cell
h^1_t
c^1_t
x_t

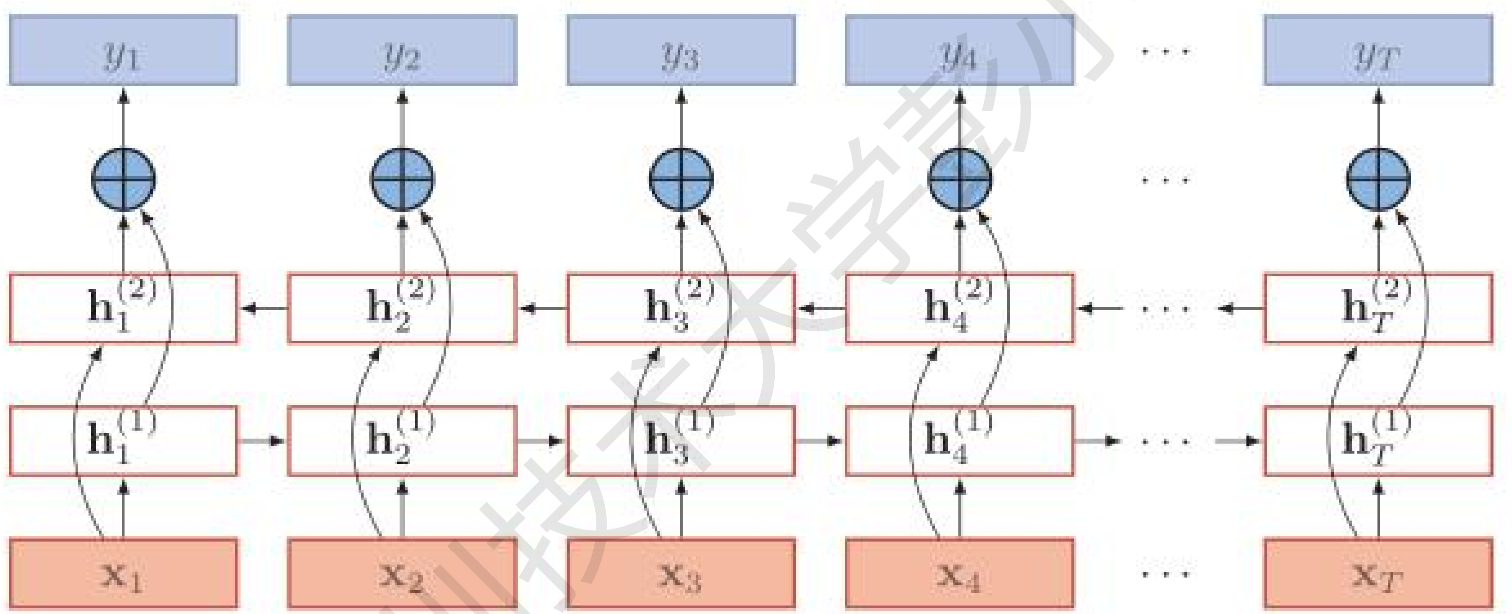
堆叠循环神经网络

- 可以是RNN, LSTM等



双向循环神经网络

- 可以是RNN, LSTM等



LSTM的Pytorch实现

- nn.LSTMCell　nn.LSTM （支持batch输入）

Pytorch LSTM

```
lstm = nn.LSTMCell(10, 20) #input_dim, recurrent dim
input = torch.randn(2, 3, 10) # (time_steps, batch, input_size)
hx = torch.randn(3, 20) # (batch, hidden_size)
cx = torch.randn(3, 20)
output = []
for i in range(input.size()[0]):
    hx, cx = lstm(input[i], (hx, cx)) # 每次输入一个时间样本
    output.append(hx)
output = torch.stack(output, dim=0)
output.shape
```

LSTM的Pytorch实现

- nn.LSTMCell   nn.LSTM （支持batch输入）

```
class lstmNet(nn.Module):
    def __init__(self, dim_input, dim_recurrent, num_layers, dim_output):
        super(lstmNet, self).__init__()
        self.lstm = nn.LSTM(dim_input, dim_recurrent, num_layers)
        self.fc = nn.Linear(dim_recurrent, dim_output)
    def forward(self, x):
        hx, cx = self.lstm(x)
        o = hx[-1, :, :]
        o = o.squeeze(axis=0)
        return self.fc(o)

input = torch.randn(2, 3, 10) #T N C
lstm = lstmNet(10, 20, 1, 30)
output = lstm(input)
output.shape
```

Gated Recurrent Unit (GRU)

- Cho等人提出的LSTM简化版本（2014）

$$r_t = \text{sigm}\left(W_{(x\ r)}x_t + W_{(h\ r)}h_{t-1} + b_{(r)}\right) \qquad \text{(reset gate)}$$

$$z_t = \text{sigm}\left(W_{(x\ z)}x_t + W_{(h\ z)}h_{t-1} + b_{(z)}\right) \qquad \text{(forget gate)}$$

$$\bar{h}_t = \tanh\left(W_{(x\ h)}x_t + W_{(h\ h)}(r_t \odot h_{t-1}) + b_{(h)}\right) \qquad \text{(full update)}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \bar{h}_t \qquad \text{(hidden update)}$$

Gated Recurrent Unit (GRU)

- Cho等人提出的LSTM简化版本（2014）

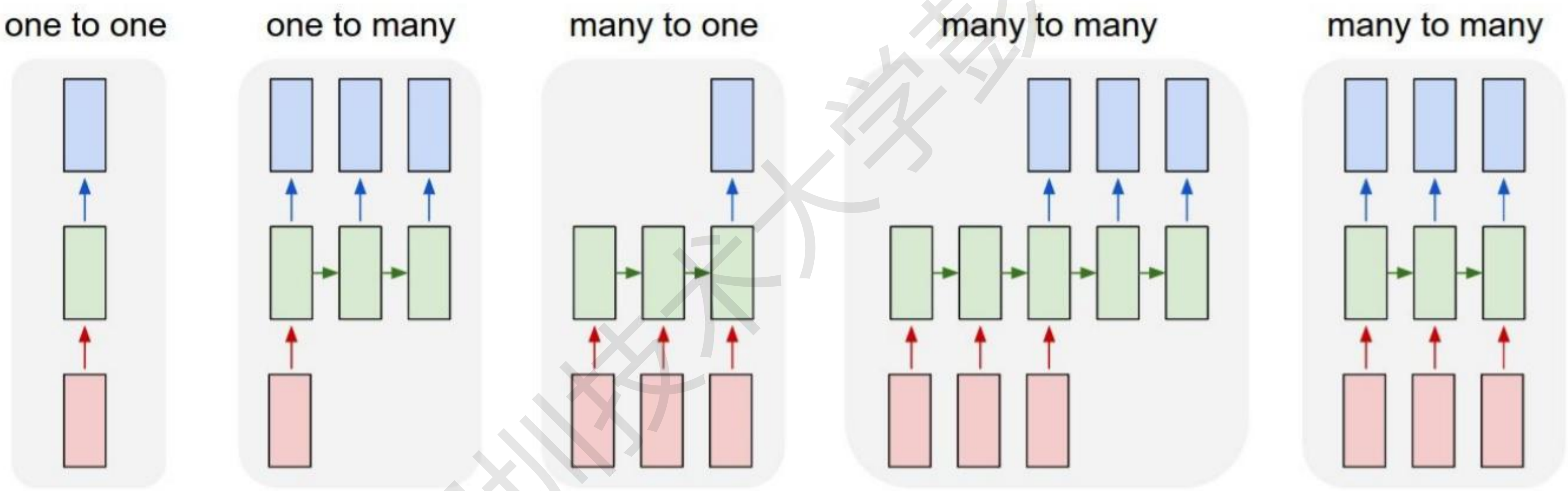
GRU

```
class gruNet(nn.Module):
    def __init__(self, dim_input, dim_recurrent, num_layers, dim_output):
        super(gruNet, self).__init__()
        self.gru = nn.GRU(dim_input, dim_recurrent, num_layers)
        self.fc = nn.Linear(dim_recurrent, dim_output)
    def forward(self, x):
        hx, cx = self.gru(x)
        o = hx[-1, :, :]
        o = o.squeeze(axis=0)
        return self.fc(o)

input = torch.randn(2, 3, 10) #T N C
lstm = lstmNet(10, 20, 1, 30)
output = lstm(input)
output.shape
```

循环网络应用

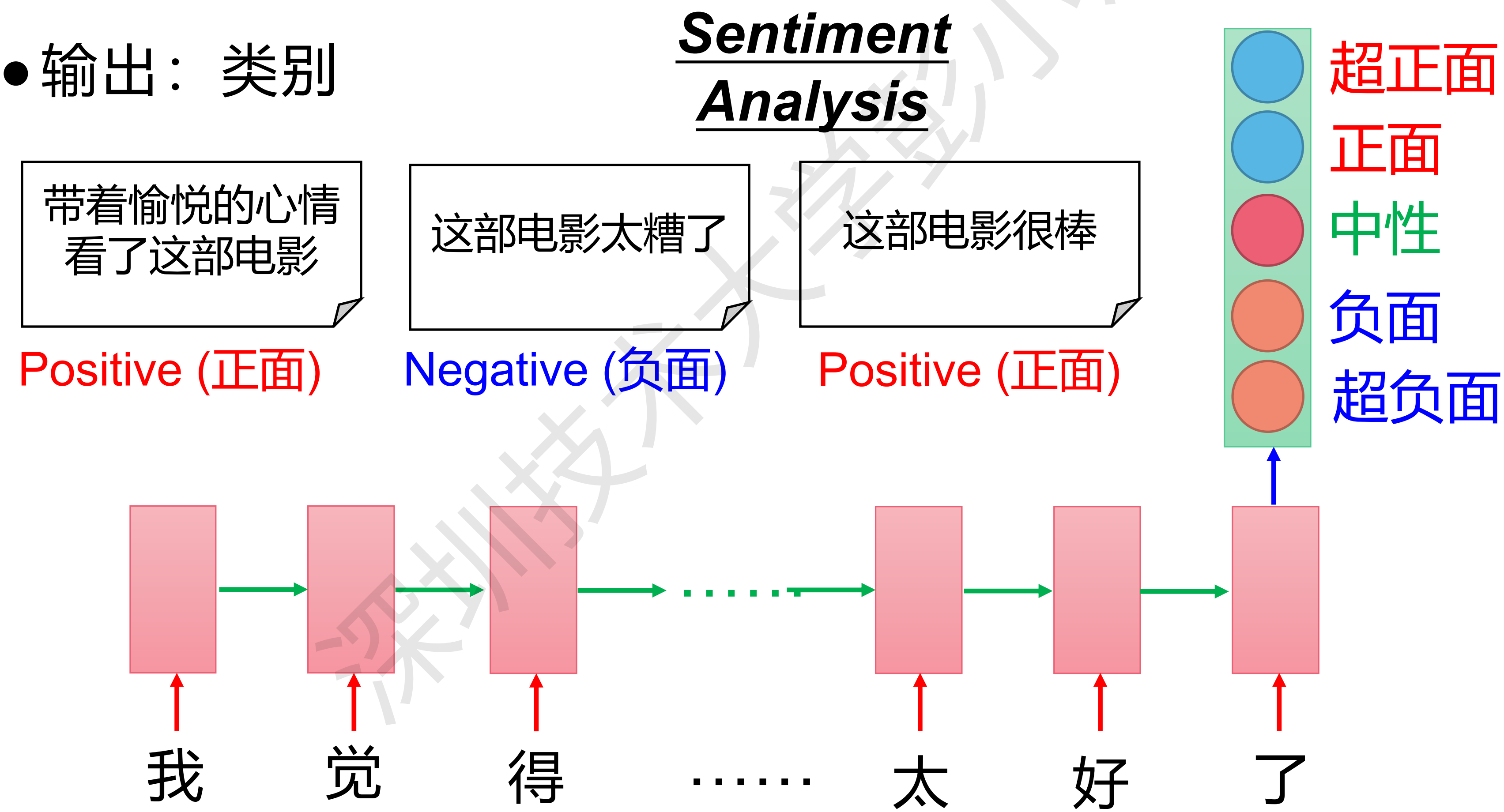
循环网络应用模式



序列到类别

来源：李宏毅《1天搞懂深度学习》

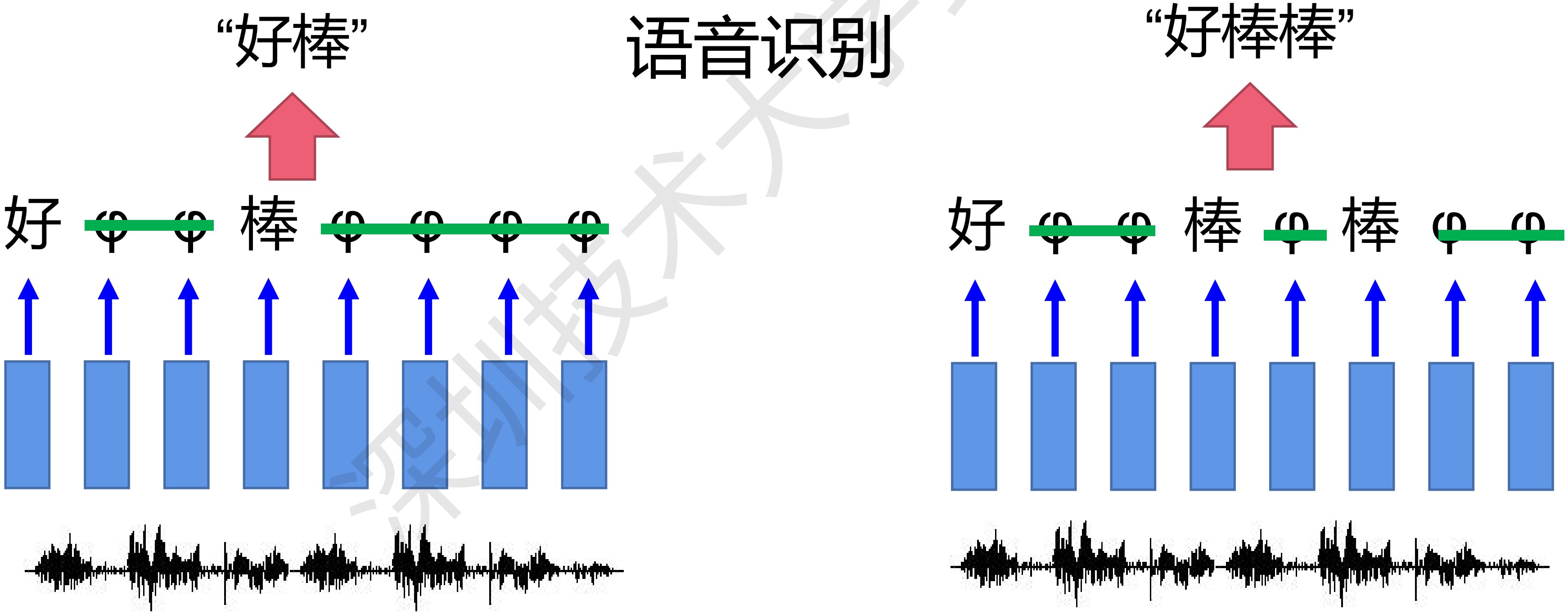
- 输入：序列
- 输出：类别



同步的序列到序列模式

来源：李宏毅《1天搞懂深度学习》

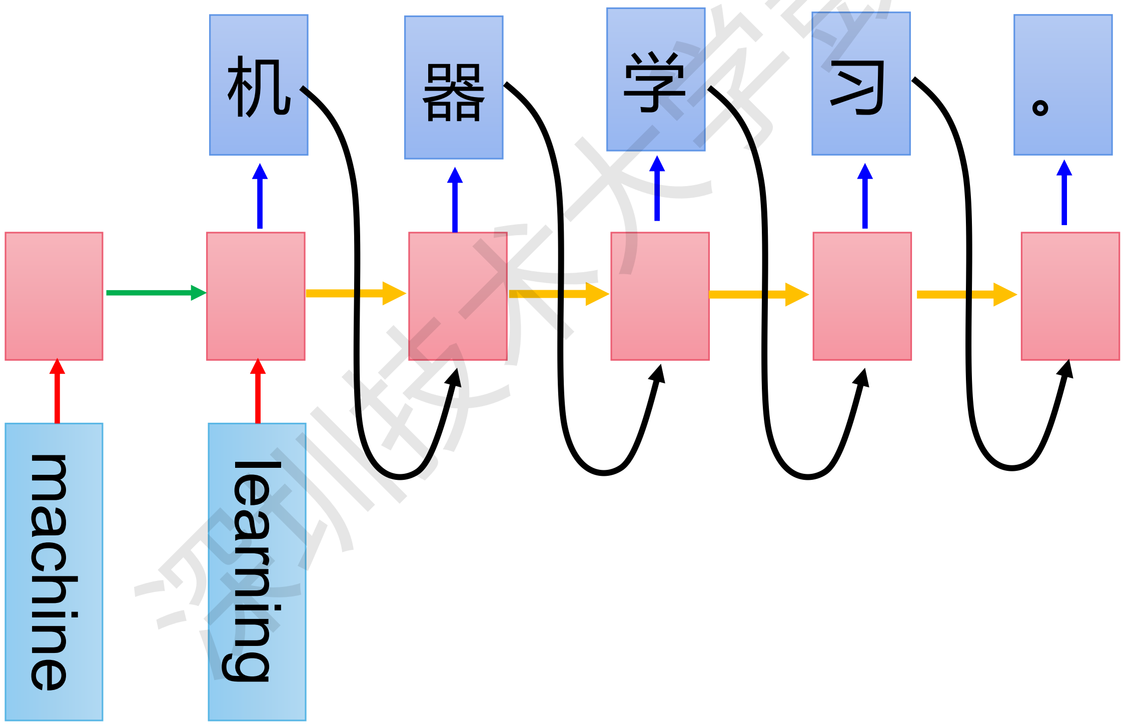
- Connectionist Temporal Classification (CTC)
  [Alex Graves, ICML’ 06][Alex Graves, ICML’ 14][Haşim Sak, Interspeech’ 15][Jie Li, Interspeech’ 15][Andrew Senior, ASRU’ 15]



异步的序列到序列模式

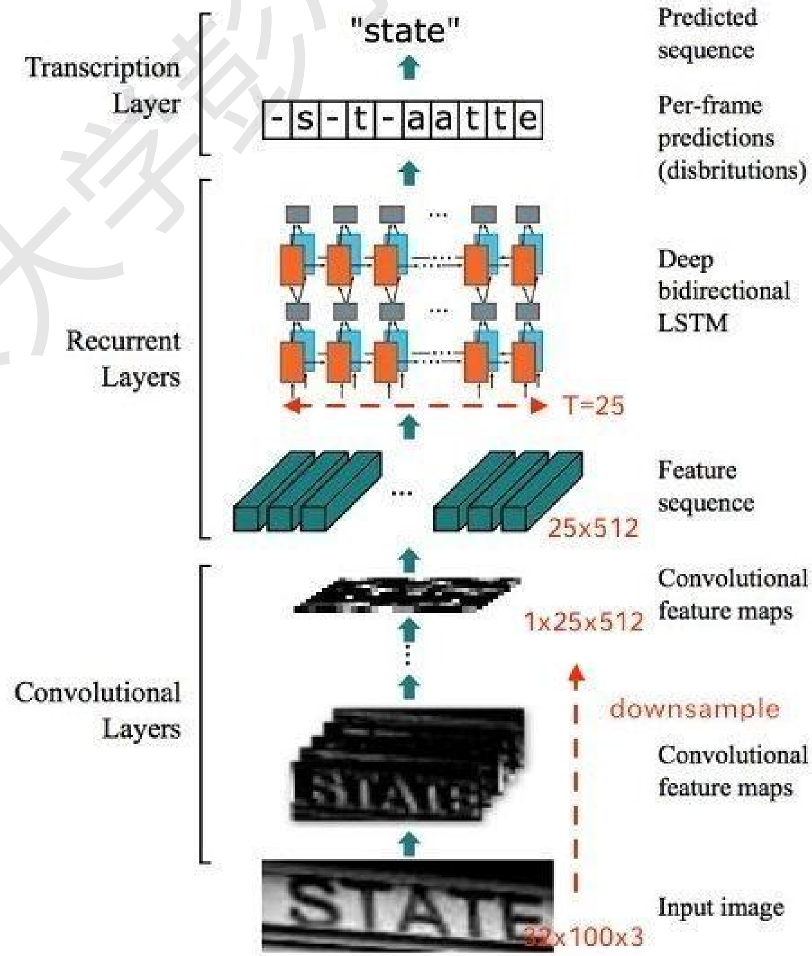
来源：李宏毅《1天搞懂深度学习》

- 机器翻译

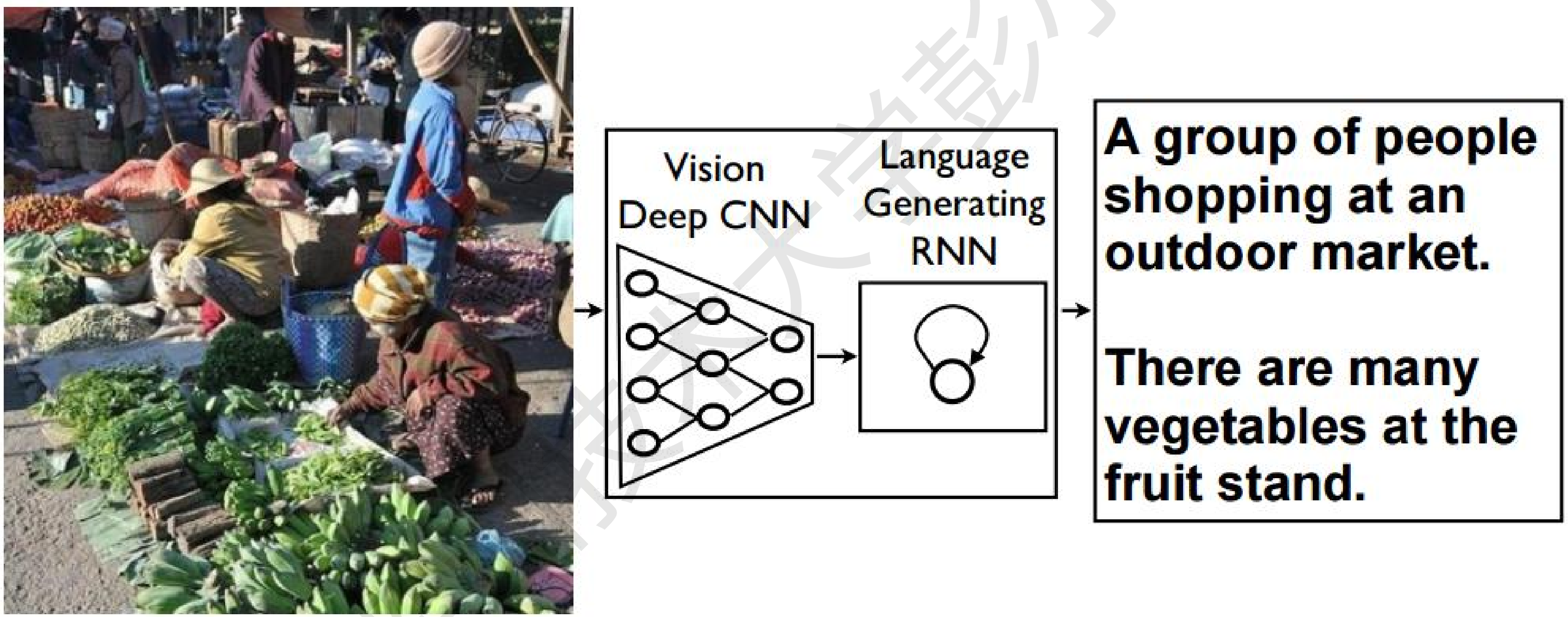


场景文字识别

- CNN+RNN+CTC(CRNN+CTC)
- 亦可用于车牌识别



看图说话

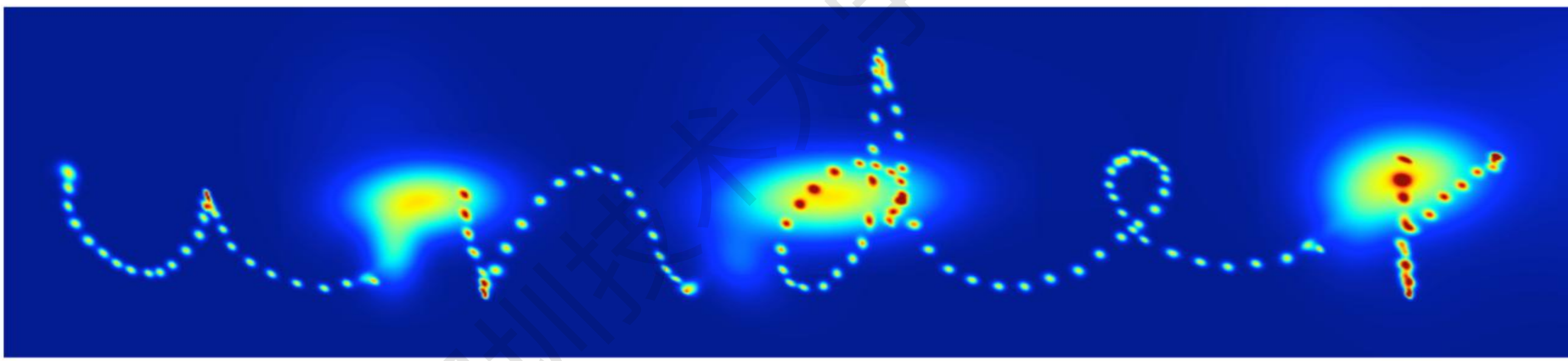

Vision
Deep CNN
Language
Generating
RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

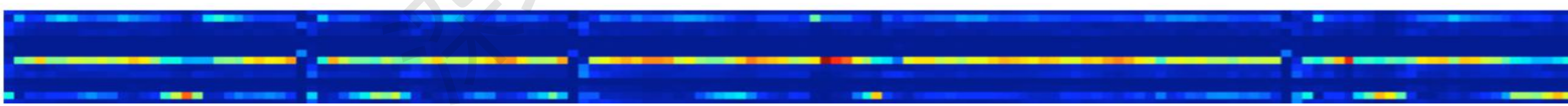
看图说话

Figure 5. A selection of evaluation results, grouped by human rating.

写字

- 把一个字母的书写轨迹看作是一连串的点。一个字母的“写法”其实是每一个点相对于前一个点的偏移量，记为(offset x, offset y)。再增加一维取值为0或1来记录是否应该“提笔”。





循环神经网络Pytorch实战1

电影评论情感分类

Pytorch实战：电影评论情感分类

- 准备IMDB数据集
  http://ai.stanford.edu/~amaas/data/sentiment/
- 实例化dataset，准备dataloader，即设计一个类来获取样本
- 构建模型，定义模型多少层、形状变化、用什么激活函数等
- 模型训练，观察迭代过程中的损失
- 模型评估，观察分类的准确率

IMDB

- 包含了5万条流行电影的评论数据，其中训练集25000条，测试集25000条
- 数据的标签以文件名的方式呈现，即序号_情感评分。情感评分中1-4为neg，5-10为pos，共有10个分类。右边为文件中的评论内容。每个文件中文本长度不一定相等，例如（pos）：

Brilliant and moving performances by Tom Courtenay and Peter Finch.

This film has its detractors, and Courtney's fey dresser may offend some folks
(who, frankly, ne
every way: enga
Courtney makes
both leads, and
is well captured,
savored like a fi

I went and saw this movie last night after being coaxed to by a few friends of mine. I'll admit that I was reluctant to see it because from what I knew of Ashton Kutcher he was only able to do comedy. I was wrong. Kutcher played the character of Jake Fischer very well, and Kevin Costner played Ben Randall with such professionalism. The sign of a good movie is that it can toy with our emotions. This one did exactly that. The entire theater (which was sold out) was overcome by laughter during the first half of the movie, and were moved to tears during the second half. While exiting the theater I not only saw many women in tears, but many full grown men as well, trying desperately not to let anyone see them crying. This movie was great, and I suggest that you go see it before you judge.

预备知识：文本的表示

- 文本是一种非常重要的非结构化的数据，如何表示文本数据一直是机器学习领域的一个重要研究方向。主流方法包括：
  1. 词袋模型
  2. tf-idf
  3. 词嵌入模型

词袋模型

- 首先了解one-hot编码，假设语料库：

  - John likes to watch movies. Mary likes too.
  - John also likes to watch football games.

- 给每个词放入字典并给与编号

  {"John": 1, "likes": 2, "to": 3, "watch": 4, "movies": 5, "also": 6, "football": 7, "games": 8, "Mary": 9, "too": 10}

- 那么每个词的one-hot向量表示如下

  John: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  likes: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
  too : [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

- 这些词向量其实就可以用于神经网络的输入

词袋模型（Bag of Words）

- 句子的表示：所有词的one-hot相加（也可看成是词频统计）

  - John likes to watch movies. Mary likes too.
  - John also likes to watch football games.

  {"John": 1, "likes": 2, "to": 3, "watch": 4, "movies": 5, "also": 6, "football": 7, "games": 8, "Mary": 9, "too": 10}

  [1, 2, 1, 1, 1, 0, 0, 0, 1, 1]
  [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]

- 词袋模型很难知道这个词是否重要
- 扩展：深度学习之前在计算机视觉领域存在10余年的视觉词袋模型热潮

TF-IDF（Term Frequency - Inverse Document Frequency）

- 用来权衡这个词对于这个文档的重要度，常用的TF-IDF来计算权重，假设t为词，d为当前文档，则t在d的表示公式为：

$$TF-IDF(t,d) = TF(t,d) \times IDF(t)$$

$$IDF(t) \ = \ log\frac{\text{语料库文档总数}}{\text{包含词}t\text{的文档数}+1}$$

- 直观的解释：如果一个单词在非常多的文章中出现，那么它可能是一个比较通用的词汇，对于区分某篇文章的特殊语义的贡献

N-gram

- TF-IDF丢失了语序的表示，只能表示出现与否，以下对于TF-IDF一样

lilei likes hanmeimei
hanmeimei likes lilei

- N-gram几个词作为item，如2-gram

{"lilei likes":1, "likes hanmeimei ":2}
{"hanmeimei likes":1, "likes lilei ":2}

- 保留了语序，但是表示向量太长

最简单的词向量表示

- 建立字典后，每个词赋予一个随机向量（本次实战采用此方式）pytorch: nn.Embedding

- 利用深度学习在海量语料库中学习的词向量代表方法：glove，word2vec

文本表示获取步骤

- 文档分词：将文本处理成一个个词（注意并不是和jieba工具包一样对中文分词）
- 长度处理：为了符合pytorch带batch的循环神经网络训练，需要统一长度，假设每条评论文本都有max_len个词，比50个词长的文本进行截取操作，比50个词短的文本则填充到50
- 词典生成：数据库所有文档的词
- 词向量生成：根据词典index赋予随机向量

IMDB+pytorch从零开始实现

```python
import torch
from torch.utils.data import DataLoader,Dataset
import os
import re

# 路径需要根据情况修改，文件太大的时候可以引用绝对路径
data_base_path = r"F:\SZTU-教学文件\UG-深度学习方法与应用\examples\data\aclImdb_v1\aclImdb"

#1. 定义tokenize的方法，对评论文本分词
def tokenize(text):
    # fileters = '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n'
    fileters = ['!','"','#','$','%','&','\(','\)','\*','\+',',','-','\.','/',':',';','<','=','>','\?','@'
        ,'\[','\\','\]','^','_','`','\{','\|','\}','~','\t','\n','\x97','\x96','”','“',]
    # sub方法是替换
    text = re.sub("<.*?>"," ",text,flags=re.S)# 去掉<...>中间的内容，主要是文本内容中存在<br/>等内容
    text = re.sub("|".join(fileters)," ",text,flags=re.S)# 替换掉特殊字符，'|'是把所有要匹配的特殊字符连在一起
    return [i.strip() for i in text.split()]# 去掉前后多余的空格


#2. 准备dataset
class ImdbDataset(Dataset):
    def __init__(self,mode):
        super(ImdbDataset,self).__init__()
        # 读取所有的训练文件夹名称
        if mode=="train":
            text_path = [os.path.join(data_base_path,i)  for i in ["train/neg","train/pos"]]
        else:
            text_path =  [os.path.join(data_base_path,i)  for i in ["test/neg","test/pos"]]

        self.total_file_path_list = []
        # 进一步获取所有文件的名称
        for i in text_path:
            self.total_file_path_list.extend([os.path.join(i,j) for j in os.listdir(i)])
```


参考：https://blog.csdn.net/Delusional/article/details/113357449

Pytorch+torchtext+LSTM

- 使用torchtext对IMDB进行预处理 pip install torchtext

..		
__init__.py	Adding Multi30k dataset (#1306)	8 months ago
ag_news.py	Fixing dataset test failures due to incorrect caching mode (#1517)	5 days ago
amazonreviewfull.py	replace funny os.sep joins with os.path.join for consistency. (#1506)	8 days ago
amazonreviewpolarity.py	replace funny os.sep joins with os.path.join for consistency. (#1506)	8 days ago
conll2000chunking.py	migrate CONLL 2000 to datapipes. (#1515)	19 hours ago
dbpedia.py	migrate DBPedia to datapipes. (#1500)	8 days ago
enwik9.py	fixing stylecheck (#1247)	11 months ago
imdb.py	fix: get sample (#1354)	7 months ago
iwslt2016.py	fixing stylecheck (#1247)	11 months ago
iwslt2017.py	fixing stylecheck (#1247)	11 months ago
multi30k.py	Update multi30k.py (#1351)	7 months ago
penntreebank.py	fixing stylecheck (#1247)	11 months ago
sogounews.py	replace funny os.sep joins with os.path.join for consistency. (#1506)	8 days ago
squad1.py	migrate SQUAD1 to datapipes. (#1513)	4 days ago
squad2.py	add initial pass at migrating SQUAD2 to datapipes. (#1514)	3 days ago
udpos.py	fixing stylecheck (#1247)	11 months ago
wikitext103.py	fixing stylecheck (#1247)	11 months ago

torchtext中的IMDB

from torchtext.legacy import data
from torchtext.legacy import datasets

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt

SEED = 1234
#
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
TEXT = data.Field(fix_length=50) #sequential=True,fix_length=50 tokeniz
# TEXT = data.Field(tokenize='spacy',tokenizer_language = 'en',
#                   include_lengths = True) #python -m spacy download e
LABEL = data.LabelField(dtype=torch.float)
train_data, test_data = datasets.IMDB.splits(TEXT, LABEL)
print(f'Number of training examples: {len(train_data)}')
print(f'Number of testing examples: {len(test_data)}')
```

循环神经网络Pytorch实战2

黑烟视频分类

黑烟视频分类（CNN+RNN）

- 连续3帧的疑似区域

主要思路

- 数据准备、格式定义等
- 编写dataset
- CNNLSTM模型定义
- 训练测试代码

黑烟视频分类（CNN+RNN）

```python
from lecture5dataset import VideoSmoke

class cnnlstm(nn.Module):
    def __init__(self, pretrained_=False):
        super(cnnlstm, self).__init__()
        self.res18 = models.resnet18(pretrained=pretrained_)
        self.res18.fc = nn.Linear(512,256)
        self.lstm = nn.LSTM(256, 256)
        self.classifier = nn.Linear(256,2)

    def forward(self, x):
        x = self.res18(x)
        # N C
        x = x.view(3,-1,256) # T N C
        y, (hx, cx) = self.lstm(x)
        x = y[-1,:,:]
        x = x.squeeze(axis=0)
        x = self.classifier(x)
        return x
```

参考文献

1. Hochreiter S， Schmidhuber J . Long Short-Term Memory[J]. Neural Computation, 1997, 9(8):1735-1780.

2. F. A. Gers, N. N. Schraudolph, and J. Schmidhuber. Learning precise timing with lstmrecurrent networks. Journal of Machine Learning Research (JMLR), 3:115–143, 2003.

3. Graves A . Long Short-Term Memory[J]. Springer Berlin Heidelberg, 2012.

4. Jozefowicz R， Zaremba W， Sutskever I . An empirical exploration of recurrent network architectures. ICML, 2015.