浙江大学

本科实验报告

课程名称：自然语言处理

姓　　名：张溢弛

学　　院：计算机科学与技术学院

专　　业：软件工程 1801

学　　号：3180103772

指导教师：汤斯亮

2021 年 5 月 31 日

浙江大学 实验报告

课程名称：自然语言处理　　实验类型：综合型

实验项目名称：实验二：基于 LSTM 的电影评论分类

学生姓名：张溢弛　专业：软件工程 1801　学号：3180103772

同组学生姓名：独立完成　指导老师：汤斯亮

实验地点：玉泉一舍 376　实验日期：2021 年 5 月 31 日

目录

一　项目介绍

1.1　实验选题

基于 LSTM 的 IMDB 数据集电影评论分类与情感分析

1.2　实验内容

基于 IMDB 数据集中的 50000 条电影评论 (训练集和测试集各 25000 条)，并使用开源的预训练词向量和 LSTM 模型进行对评论进行情感分析和文本分类，并调整参数进行探究。同时我也自己按照网络上公开的教程使用 Pytorch 搭建了同样的模型，并和运行在华为云平台上的基于 mindspore 框架的同类型模型进行性能的对比。

1.3　实验环境

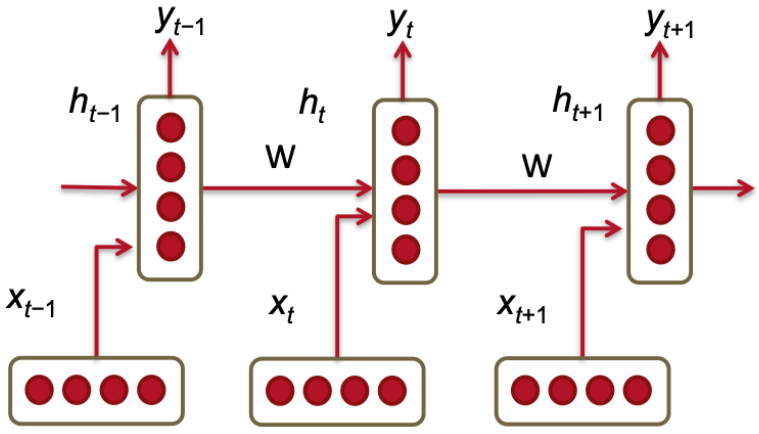
- 操作系统：华为云 ModelArts 云平台
- 编程环境：Python3+Jupyter Notebook
- 使用的 Python 库：mindspore1.1.1

二　技术细节

2.1　技术原理

2.1.1　RNN

循环神经网络 (Recurrent Neural Networks) 是可以根据语料库中所有之前出现过的额单词来调整模型，RNN 的基本组成单元如下图所示：



每个神经网络单元 (也称为 timestep) 会对输入的词嵌入向量 x_t 进行一个矩阵运算，并且

对上一个位置的单元传递下来的 h_{t-1} 进行运算以后相加得到 h_t 传递给下一个单元，然后对 h_t 使用非线性函数激活和 sotfmax 函数之后输出一个概率分布 $\hat{y}_t$ 作为当前单元的输出：

$$\begin{aligned} h_t &= \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right) \\ \hat{y}_t &= \text{softmax}\left(W^{(S)}h_t\right) \end{aligned} \tag{1}$$

并且权重矩阵 $W^{(hh)}, W^{(hx)}$ 在不同的 timestep 之间时共享的，这样一来**一个 RNN 模型需要学习的参数数量是比较少的，并且和输入语料库的长度是没有无关的**，不仅避免了参数的维度爆炸，而且可以处理任意长度的输入结果。RNN 的损失函数通常采用交叉熵来计算，对于第 t 个 timestep，其交叉熵可以表示为：

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \times \log\left(\hat{y}_{t,j}\right) \tag{2}$$

这样一来总的交叉熵就可以表示为：

$$J(\theta) = \frac{1}{T}\sum_{i=1}^{T} J^{(t)}(\theta) = -\frac{1}{T}\sum_{i=1}^{T}\sum_{j=1}^{|V|} y_{t,j} \times \log\left(\hat{y}_{t,j}\right) \tag{3}$$

RNN 的优点主要有如下几个：

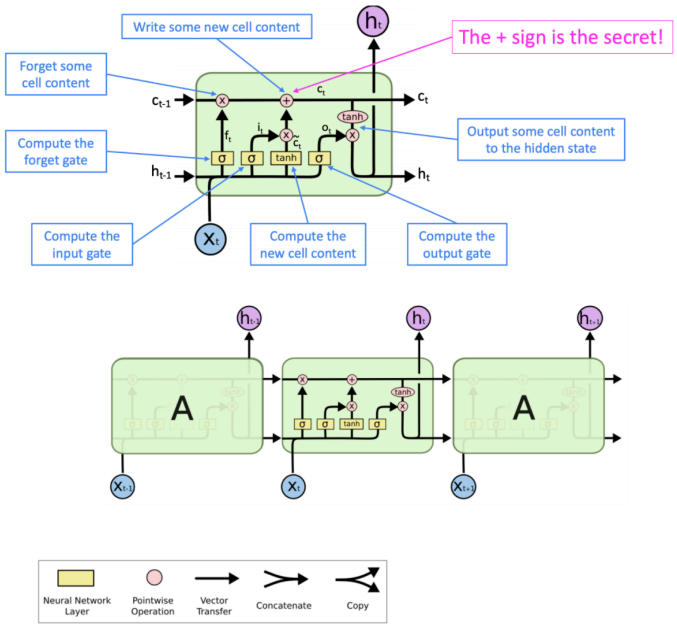
- RNN 可以处理任意长度的输入
- RNN 模型的 size 不会随着输入的语料库长度变化而变化
- 某个 timestep 计算得到的结果理论上可以在后面的很多 timestep 中被用到
- 因为每个 timestep 的权重是相同的，因此 RNN 模型具有一定的对称性

但是同时，RNN 的缺点也是很明显的：

- RNN 是一个序列化的模型，因此 **RNN 不能进行并行的训练**
- 事实上随着 timestep 的推移，前面的信息越来越难以被保留下来，可能存在**梯度消失和梯度爆炸**的问题

2.1.2 LSTM

LSTM 的全称是长短期记忆 (Long Short Term Memory)，是用于保存长期记忆的一种复杂激活单元架构，如下图所示：



这种架构下，隐藏层中主要进行的数学运算有：

$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right) \quad \text{(Input gate)} \\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right) \quad \text{(Forget gate)} \\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right) \quad \text{(Output/Exposure gate)} \\
\tilde{c}_t &= \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right) \quad \text{(New memory cell)} \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \quad \text{(Final memory cell)} \\
h_t &= o_t \circ \tanh\left(c_t\right)
\end{aligned}
\tag{4}
$$

LSTM 中有若干个门，分别叫做输入门，输出门和遗忘门，起到一定的判断作用，用来判断 LSTM 中的一些信息是否值得保留，LSTM 的计算过程可以描述为：

- 生成新记忆：根据上一个单元传入的 h_{t-1} 和当前位置的词向量 x_t 计算出一个结果 $\tilde{c}_t$
- 输入门：根据上一个单元传入的 h_{t-1} 和当前位置的词向量 x_t 计算出一个结果 i_t，并评估当前位置的单词 x_t 是否有用，并有必要在之后的计算中使用
- 遗忘门：根据上一个单元传入的 h_{t-1} 和当前位置的词向量 x_t 计算出一个结果 f_t，并且评估上一个隐藏状态 h_{t-1} 是否有价值用在当前层的计算中
- 生成最终记忆：根据输入门和遗忘门的判断，并结合新的记忆单元和上一个传递过来的记忆单元的信息生成最终记忆
- 输出门：这是一个 GRU 中不存在的门，其目的是将最终的记忆单元和隐藏状态分开，因为最终生成的记忆单元有很多前文的信息是没有必要保存在当前隐藏状态中的，因此这一个单元决定了最终记忆 c_t 有多少内容需要暴露给 h_t，根据这个权重来产生最终的隐藏状态

2.2 实验细节与关键步骤

本次实验的细节和关键的代码主要分为如下几个部分，其中实验结果将在第三部分具体给出。

2.2.1 实验环境配置

注册华为云平台的账号，由于代金券还没有到账因此自费充值来使用 ModelArts 平台，按照教程中创建好 OBS 桶并在 ModelArts 中配置好实验环境之后开始实验，其中数据集需要通过 moxing 库从 OBS 桶中复制到云平台当前的工作目录下。

2.2.2 代码文件结构

本次实验的基本文件结构如下图所示：

```
├── aclImdb
│   ├── imdbEr.txt
│   ├── imdb.vocab
│   ├── README
│   ├── test
│   └── train
├── glove
│   ├── glove.6B.50d.txt
│   ├── glove.6B.100d.txt
│   ├── glove.6B.200d.txt
│   └── glove.6B.300d.txt
├── src
│   ├── config.py
│   ├── lstm.py
│   ├── imdb.py
│   ├── lr_schedule.py
│   └── dataset.py
└── main.py
```

2.2.3 模型文件结构

其中 main.py 是代码的运行的入口文件，而 src 目录下包含了 LSTM 模型的关键代码，dataset.py 和 imdb.py 文件是用来对 imdb 数据集进行预处理，并使之生成对应的 MindRecord 结构的数据，可以被 mindspore 框架正常处理，而 config.py 包含了模型训练参数的配置，lr_schedule 是当使用华为的 Ascend 加速的时候使用的自适应学习率生成器，用于在训练中调整合适的学习率 (不过这一部分功能在正式训练中似乎没有用到，因为我在云平台中使用了 CPU 进行训练)，而 lstm.py 是最关键的，实现了层级化的 LSTM 和 SentimentNet 等实验所需的关键模型，其中的 mindspore 模型构建是非常值得学习的。

2.2.4 数据集

本实验使用的是 imdb 数据集，在目录下的 mdb.vocab 包含了一个词汇表，而 train 和 test 目录下面分别包含了训练集和数据集，分别由 25000 条电影评论构成，此外实验中还是用了已经完成预训练的 glove 词向量，这些训练好的词向量位于 glove 目录下，有 50 维，100 维，200 维，300 维四种选择，但是在正式训练的时候因为算力原因，出于节约代金券的目的我使用了 50 维的词向量，因此最终的训练效果相比于教程里的也会略差。

2.2.5 数据预处理

在开始预处理之前，我们要安装 gensim 库才能从 glove 文件中正确读取词向量：

```
! pip install gensim
```

```
Looking in indexes: http://repo.myhuaweicloud.com/repository/pypi/simple
Collecting gensim
  Downloading http://repo.myhuaweicloud.com/repository/pypi/packages/1f/6c/363d00aa23642f42b27b908c6474ab981c75882eefc084210d5b8ce8cd8e/gensim-4.0.1.tar.gz (23.1MB)
     100% |████████████████████████████████| 23.1MB 70.5MB/s ta 0:00:01
Requirement already satisfied: numpy>=1.11.3 in /home/ma-user/miniconda3/envs/Mindspore-python3.7-aarch64/lib/python3.7/site-packages (from gensim) (1.17.5)
Requirement already satisfied: scipy>=0.18.1 in /home/ma-user/miniconda3/envs/Mindspore-python3.7-aarch64/lib/python3.7/site-packages (from gensim) (1.5.4)
Collecting smart_open>=1.8.1 (from gensim)
  Downloading http://repo.myhuaweicloud.com/repository/pypi/packages/2f/36/5f7cdd039b2a49636aa098145a3f233601d4580805ccf19819743bd7dad1/smart_open-5.0.0-py3-none-any.whl (56kB)
     100% |████████████████████████████████| 61kB 22.6MB/s ta 0:00:01
Building wheels for collected packages: gensim
  Running setup.py bdist_wheel for gensim ... done
  Stored in directory: /home/ma-user/.cache/pip/wheels/c5/e4/1a/878be388fbd01b972201c6a184063eb1fb7f65bf1f5de23114
Successfully built gensim
```

然后在 jupyter notebook 中输入命令行就可以运行 main.py 文件进行预处理，第一次训练的时候将 preprocess 参数设置为 true 就可以进行预处理。

```
! python main.py --preprocess=true --data_url=./aclImdb --glove_path=./glove --ckpt_path=./ckpt --train_url=./train --preprocess_path=./
```

```
[WARNING] ME(7788:281473872419120,MainProcess):2021-05-24-13:41:46.432.962 [mindspore/_check_version.py:207] MindSpore version 1.1.1 and "te" wheel package version 1.0 does not match, reference to the match info on: https://www.mindspore.cn/install
MindSpore version 1.1.1 and "topi" wheel package version 0.6.0 does not match, reference to the match info on: https://www.mindspore.cn/install
WARNING: 'ControlDepend' is deprecated from version 1.1 and will be removed in a future version, use 'Depend' instead.
[WARNING] ME(7788:281473872419120,MainProcess):2021-05-24-13:41:46.957.893 [mindspore/ops/operations/array_ops.py:2302] WARN_DEPRECATED: The usage of Pack is deprecated. Please use Stack.
/home/ma-user/miniconda3/envs/Mindspore-python3.7-aarch64/lib/python3.7/site-packages/gensim/similarities/__init__.py:15: UserWarning: The gensim.similarities.levenshtein submodule is disabled, because the optional Levenshtein package <https://pypi.org/project/python-Levenshtein/> is unavailable. Install Levenhstein (e.g. `pip install python-Levenshtein`) to suppress this warning.
  warnings.warn(msg)
============== Starting Data Pre-processing ==============
============== Starting Training ==============
```

预训练完成之后会在 prepreprocess 目录下生成一系列数据集文件。

2.2.6 模型的训练

在训练模型的时候我很神奇的发现华为云平台 ModelArts 创建的基于 Ascend 加速的 mindspore 虚拟环境不支持对 LSTM 算子的加速，刚开始尝试的时候一直会报出各种各样的错误，查了一下好像确实如此，所以我就把代码中所有的相关配置项都改成了 CPU 才让训练正常进行下去，而因为使用了 CPU 所以训练效率比较低下，因此我只能选择了参数量比较少的的 50 维词向量，并修改了 batch size 参数和 epoch 数，以减少训练次数，最终我进行了两次比较完整的训练，其关键参数的配置如下：

- 词向量 50 维，batch size=5000，epoch=1(尝试性训练一个模型)
- 词向量 50 维，batch size=500，epoch=5

这里实在是因为华为云平台的算力有限，所以才简单了训练了几个比较快速的模型 (实际上还是花了 10 个小时左右的时间才练出两个能看的模型，因为前面一直在试错，出现了各种各样的问题)，不过在那之后我尝试自己使用 Pytorch 搭建了一个 LSTM 模型来实现 IMDB 数据集下的分类任务，具体的在下一节中讲。

2.3 自己的尝试

我参考 Github 上随处可见的教程，使用 Pytorch 搭建了一个 LSTM 模型来尝试对 IMDB 数据集进行情感分类，具体的代码可以在提交的压缩包中看到，这里我使用了 2 层 LSTM 和 2 个全连接层搭建了模型，跟实验提供的代码的模型有所区别，并完成了 IMDB 数据集的预处理和训练，测试等代码，模型的架构如下图所示：

```
NaiveLSTM(
  (Embedding): Embedding(10000, 128)
  (LSTM): LSTM(128, 128, num_layers=2, batch_first=True, bidirectional=True)
  (dp): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=256, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=2, bias=True)
)
```

最后训练得到的结果发现比在华为云平台上要好不少，当然和参数设置、模型架构也有比较大的关系，但不得不说 Pytorch 还是比上面用的 ModelArts 和 mindspore 好用太多。

三 实验结果

3.1 第一次训练

第一次训练使用的参数是：词向量 50 维，batch size=5000，epoch=1，因为好不容易成功之后想快速得到一个模型看看效果，最终十分钟左右完成了训练得到的结果如下：

```
! python main.py --preprocess=false --data_url=./aclImdb --glove_path=./glove --train_url=./train --preprocess_path=./preprocess --devic
[WARNING] ME(161409:281473095264560,MainProcess):2021-05-24-17:24:15.128.60 [mindspore/_check_version.py:207] MindSpore version 1.1.1 an
d "te" wheel package version 1.0 does not match, reference to the match info on: https://www.mindspore.cn/install
MindSpore version 1.1.1 and "topi" wheel package version 0.6.0 does not match, reference to the match info on: https://www.mindspore.cn/
install
WARNING: 'ControlDepend' is deprecated from version 1.1 and will be removed in a future version, use 'Depend' instead.
[WARNING] ME(161409:281473095264560,MainProcess):2021-05-24-17:24:15.526.054 [mindspore/ops/operations/array_ops.py:2302] WARN_DEPRECATE
D: The usage of Pack is deprecated. Please use Stack.
/home/ma-user/miniconda3/envs/Mindspore-python3.7-aarch64/lib/python3.7/site-packages/gensim/similarities/__init__.py:15: UserWarning: T
he gensim.similarities.levenshtein submodule is disabled, because the optional Levenshtein package <https://pypi.org/project/python-Leve
nshtein/> is unavailable. Install Levenhstein (e.g. `pip install python-Levenshtein`) to suppress this warning.
  warnings.warn(msg)
============== Starting Training ==============
epoch: 1 step: 1, loss is 0.69336337
epoch: 1 step: 2, loss is 0.6932762
epoch: 1 step: 3, loss is 0.6933262
epoch: 1 step: 4, loss is 0.6932817
epoch: 1 step: 5, loss is 0.69323313
epoch time: 195943.993 ms, per step time: 39188.799 ms
============== Training Success ==============
============== Starting Testing ==============
============== {'acc': 0.48324} ==============
```

最终结果不太好，可以看到 loss 函数降不下去，准确率也比较低

3.2 第二次训练

第二次训练使用的参数是：词向量 50 维，batch size=500，epoch=5，这一次花了比较久的时间，大约 40 分钟把训练完成，结果如下：

```
! python main.py --preprocess=false --data_url=./aclImdb --glove_path=./glove --train_url=./train --preprocess_path=./preprocess --devic
TED: The usage of Pack is deprecated. Please use Stack.
/home/ma-user/miniconda3/envs/Mindspore-python3.7-aarch64/lib/python3.7/site-packages/gensim/similarities/__init__.py:15: UserWarning:
The gensim.similarities.levenshtein submodule is disabled, because the optional Levenshtein package <https://pypi.org/project/python-L
evenshtein/> is unavailable. Install Levenhstein (e.g. `pip install python-Levenshtein`) to suppress this warning.
  warnings.warn(msg)
============== Starting Training ==============
epoch: 1 step: 1, loss is 0.6926479
epoch: 1 step: 2, loss is 0.6942082
epoch: 1 step: 3, loss is 0.6931035
epoch: 1 step: 4, loss is 0.69283193
epoch: 1 step: 5, loss is 0.69343203
epoch: 1 step: 6, loss is 0.69316554
epoch: 1 step: 7, loss is 0.69355065
epoch: 1 step: 8, loss is 0.6929146
epoch: 1 step: 9, loss is 0.69246125
epoch: 1 step: 10, loss is 0.69376063
epoch: 1 step: 11, loss is 0.69360864
epoch: 1 step: 12, loss is 0.69499636
epoch: 1 step: 13, loss is 0.692988
```

可以看到在增大 epoch 数量，并且适当减小 batch size 之后，训练的成功率略有提高，但仍然是比较低的水平，教程中给出的准确度约为 84%

```
epoch: 3 step: 39, loss is 0.6894872
epoch: 3 step: 40, loss is 0.69277215
epoch: 3 step: 41, loss is 0.68646306
epoch: 3 step: 42, loss is 0.68886
epoch: 3 step: 43, loss is 0.68939596
epoch: 3 step: 44, loss is 0.6918189
epoch: 3 step: 45, loss is 0.6851985
epoch: 3 step: 46, loss is 0.68537223
epoch: 3 step: 47, loss is 0.69208264
epoch: 3 step: 48, loss is 0.69050926
epoch: 3 step: 49, loss is 0.6938694
epoch: 3 step: 50, loss is 0.6940223
epoch time: 757016.622 ms, per step time: 15140.332 ms
============== Training Success ==============
============== Starting Testing ==============
============== {'acc': 0.52428} ==============
```

按照词向量 300 维，batch size=32，epoch=10 训练非常久的时间之后可以得到如下效果：

```
Epoch time: 27545.180, per step time: 70.629
epoch: 9 step: 78, loss is 0.27715153
epoch: 9 step: 156, loss is 0.085485235
epoch: 9 step: 234, loss is 0.35549596
epoch: 9 step: 312, loss is 0.1265975
epoch: 9 step: 390, loss is 0.081303015
Epoch time: 27582.971, per step time: 70.726
epoch: 10 step: 78, loss is 0.19696395
epoch: 10 step: 156, loss is 0.03179455
epoch: 10 step: 234, loss is 0.11651886
epoch: 10 step: 312, loss is 0.050257515
epoch: 10 step: 390, loss is 0.025655827
Epoch time: 27546.935, per step time: 70.633
============== Training Success ==============
```

```
============== Starting Testing ==============
============== {'acc': 0.8476362179487179} ==============
```

3.3 自己模型的训练

而我自己搭建的模型在本地电脑上跑了 2 个 epoch 就达到了 80% 的准确率，并且 loss 很快降了下去，过程中的截图如下：

```
Train Epoch: 1 [4500/25000 (18%)]   Loss: 0.692040
Train Epoch: 1 [9500/25000 (38%)]   Loss: 0.649837
Train Epoch: 1 [14500/25000 (58%)]  Loss: 0.637147
Train Epoch: 1 [19500/25000 (78%)]  Loss: 0.609677
Train Epoch: 1 [24500/25000 (98%)]  Loss: 0.554742

Test set: Average loss: 0.5953, Accuracy: 17312/25000 (69%)
acc is: 0.6925, best acc is 0.6925

Train Epoch: 2 [4500/25000 (18%)]   Loss: 0.518529
Train Epoch: 2 [9500/25000 (38%)]   Loss: 0.452636
Train Epoch: 2 [14500/25000 (58%)]  Loss: 0.473183
Train Epoch: 2 [19500/25000 (78%)]  Loss: 0.468175
Train Epoch: 2 [24500/25000 (98%)]  Loss: 0.460369

Test set: Average loss: 0.4485, Accuracy: 19979/25000 (80%)
acc is: 0.7992, best acc is 0.7992
```

最终训练了 5 个 epoch 的模型的准确度就达到了 83% 左右，训练时间相比于 Mindspore 框架和 ModelArts 也有比较大的提升。

```
NaiveLSTM(
  (Embedding): Embedding(10000, 128)
  (LSTM): LSTM(128, 128, num_layers=2, batch_first=True, bidirectional=True)
  (dp): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=256, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=2, bias=True)
)

Test set: Average loss: 0.3840, Accuracy: 20762/25000 (83%)

Process finished with exit code 0
```

四　总结思考

虽然本次实验不需要我们亲自动手使用 mindspore 框架完成代码的编写，因此代码的实践比较少，而配置运行环境时踩的坑比较多，本次实验中我主要有如下收获：

- 由于华为云提供的代金券使用额度十分有限，为了节约时间只能对作业提供的代码进行简单尝试，没有涉及大规模的调参测试，因此训练的效果也不是很好，但是我使用自己搭建的 LSTM 模型取得了很好的效果，相比于实验提供的平台和代码，准确率和

训练速度都有了比较大的提高。

- ModelArts 平台和 Mindspore 框架的版本兼容性做的极差，比如本次实验中要运行的 LSTM 单元在华为提供的 Ascend 环境下时不支持的，这导致我刚运行代码的时候出现了一堆各种各样的报错，搞得我心态也十分爆炸，一度想退课，但是因为错过了退课时间只能作罢。
- 建议以后该课程的实验不要用华为云的平台，我认为这个平台还只是一个 toy demo，完全没有达到可以投入商业运营的程度，另外直接跑现成的代码也无助于学生掌握课上所教的模型和知识，只是单纯地和 Mindspore 这个莫名其妙的环境斗智斗勇。

参考文献

[1] http://web.stanford.edu/class/cs224n/index.html

[2] https://github.com/mindspore-ai/course/tree/master/lstm