作业 3：依赖解析

1. 机器学习与神经网络(8 分)

(a) Adam 优化器

Question 1(2 分): 请用 2~4 句话简要解释使用动量 **m** 是如何防止更新量过大的以及为什么小变化量从整体上讲对学习是有益的？（无需数学证明，提供直觉性的解释即可）

$$\mathbf{m}_{t+1} \leftarrow \beta_1 \mathbf{m}_t + (1-\beta_1)\nabla_{\boldsymbol{\theta}_t} J_{\text{minibatch}}(\boldsymbol{\theta}_t)$$
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \mathbf{m}_{t+1}$$

My answer: m_{t+1}是历史梯度信息m_t和当前梯度信息$\nabla_{\boldsymbol{\theta}_t} J_{minibatch}(\boldsymbol{\theta}_t)$关于参数$\beta_1$的加权求和；$\beta_1$是 0 到 1 之间的超参数，通常设成 0.9，表明历史梯度信息对更新量的影响远大于当前梯度信息，防止因当前梯度过大导致更新量过大；较小的参数更新量使得模型更稳健地收敛到最优。

Question 2(2 分): 由于 Adam 将更新量除以$\sqrt{\boldsymbol{v}}$，哪些模型参数将得到更大的更新量？为什么这有利于学习？

$$\mathbf{m}_{t+1} \leftarrow \beta_1 \mathbf{m}_t + (1-\beta_1)\nabla_{\boldsymbol{\theta}_t} J_{\text{minibatch}}(\boldsymbol{\theta}_t)$$
$$\mathbf{v}_{t+1} \leftarrow \beta_2 \mathbf{v}_t + (1-\beta_2)(\nabla_{\boldsymbol{\theta}_t} J_{\text{minibatch}}(\boldsymbol{\theta}_t) \odot \nabla_{\boldsymbol{\theta}_t} J_{\text{minibatch}}(\boldsymbol{\theta}_t))$$
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \mathbf{m}_{t+1} / \sqrt{\mathbf{v}_{t+1}}$$

My answer: 梯度值小的模型参数更新量更大；防止模型参数变化得过快或过慢，对提高学习稳健性和加速算法收敛有益。

(b) Dropout

Question 1(2 分): γ必须等于什么(用p_{drop}表示)？简要证明给出的答案或者使用上述方程给出数学推导。

$$\mathbf{h}_{\text{drop}} = \gamma \mathbf{d} \odot \mathbf{h}$$

$$\mathbb{E}_{p_{\text{drop}}}[\mathbf{h}_{\text{drop}}]_i = h_i$$

My answer:

$$\gamma \cdot [p_{drop} \cdot 0 + (1 - p_{drop}) \cdot h_i] = h_i$$

$$\gamma = \frac{1}{1 - p_{drop}}$$

Question 2(2 分): 为什么暂退法应该在训练时使用？为什么暂退法不应该在评估时使用？

My answer:

训练时使用暂退法可以抑制过拟合；评估时使用暂退法使得评估结果不稳定。

2. 基于状态转移的神经网络依赖解析(46 points)

(a) 请写出解析句子"*I attended lectures in the NLP class.*"所需的转移序列。(4 points)

Stack	Buffer	New dependency	Transition
[ROOT]	[I, attended, lectures, in, the, NLP, class]		Initial Configuration
[ROOT, I]	[attended, lectures, in, the, NLP, class]		SHIFT
[ROOT, I, attended]	[lectures, in, the, NLP, class]		SHIFT
[ROOT, attended]	[lectures, in, the, NLP, class]	attended → I	LEFT-ARC
[ROOT, attended, lectures]	[in, the, NLP, class]		SHIFT
[ROOT, attended]	[in, the, NLP, class]	attended → lectures	RIGHT-ARC
[ROOT, attended, in]	[the, NLP, class]		SHIFT
[ROOT, attended, in, the]	[NLP, class]		SHIFT
[ROOT, attended, in, the, NLP]	[class]		SHIFT
[ROOT, attended, in, the, NLP, class]	[]		SHIFT
[ROOT, attended, in, the, class]	[]	class → NLP	LEFT-ARC
[ROOT, attended, in, class]	[]	class → the	LEFT-ARC
[ROOT, attended, class]	[]	class → in	LEFT-ARC
[ROOT, attended]	[]	attended → class	RIGHT-ARC
[ROOT]	[]	ROOT → attended	RIGHT-ARC

(b) 一个包含 n 个单词的句子需要多少步完成解析，给出关于 n 的表达式，用 1-2 句话简单解释一下原因 (2 points)

My answer:

解析步数为 2n；n 个单词共需要 n 步 SHIFT 从 buffer 转移到 stack，LEFT-ARC 或 RIGHT-ARC 使 stack 中的单词数减 1，因此共需要 n 步 LEFT-ARC 或 RIGHT-ARC 使 stack 中只剩下 ROOT，此时解析完成。

(c) 实现 parser_transitions.py 中 PartialParse 类的__init__函数和 parse_step 函数。这将实现你的解析器的运行规则。你可以通过 python parser_transitions.py part_c 运行 basic 测试。(6 points)

```
(cs224n) PS F:\CV\NLP\coding\a3\student> python parser_transitions.py part_c
F:\CV\NLP\coding\a3\student\parser_transitions.py:183: SyntaxWarning: "is" with a literal. Did you mean "=="?
  return [("RA" if pp.stack[1] is "right" else "LA") if len(pp.buffer) == 0 else "S"
SHIFT test passed!
LEFT-ARC test passed!
RIGHT-ARC test passed!
parse test passed!
```

(d) 我们的网络将预测应用于部分解析的下一个转移。我们可以通过预测转移来解析单个句子，直至解析完成。然而，当一次性处理批量的数据时，神经网络的运行效率更高（即同时预测多个不同部分解析的下一个转移）。我们可以使用以下算法以小批量的方式解析句子。(8 points)

算法 1 小批量依赖解析

输入：sentences，待解析的语句列表；model，产生解析决策的模型

初始化 partial_parses 为 PartialParses 的列表，每个 PartialParses 对应 sentences 中的一个句子

初始化 unfinished_parses 为 partial_parses 的浅拷贝

while unfinished_parses 非空 **do**

将未解析的第一个批量大小的对象作为一个小批量

使用模型为小批量中的每一个部分解析的对象预测下一步转换动作

根据预测的转换动作在小批量中的每个部分解析对象上执行一步解析

从部分解析的对象中去除完全解析的对象(buffer 为空且 stack 的大小为 1)

end while

返回：partial_parses 中每个解析项的依赖结果 dependencies

Implement this algorithm in the minibatch parse function in parser transitions.py. You can run basic (non-exhaustive) tests by running python parser transitions.py part d. Note: You will need minibatch parse to be correctly implemented to evaluate the model you will build in part (e). However, you do not need it to train the model, so you should be able to complete most of part (e) even if minibatch parse is not implemented yet.

在 parser_transitions.py 中实现 minibatch_parse 算法，可以通过命令行执行 python parser_transitions.py part_d 进行基础测试。你需要正确地实现 minibatch_parse 以评估(e)中你构建的模型。然而，你并不需要训练这个模型，因此你应该也能完成(e)中的大部分工作即使 minibatch_parse 尚未实现。

```
(cs224n) PS F:\CV\NLP\coding\a3\student> python parser_transitions.py part_d
F:\CV\NLP\coding\a3\student\parser_transitions.py:202: SyntaxWarning: "is" with a literal. Did you mean "=="?
  return [("RA" if pp.stack[1] is "right" else "LA") if len(pp.buffer) == 0 else "S"
minibatch_parse test passed!
```

(e) 我们现在要训练一个神经网络，给网络提供栈、缓冲区和依赖关系的状态，由网络完成预测，随后进行状态转移。首先，模型提取表示当前状态的一个特征向量。我们将使用最初的神经依赖解析论文"*A Fast and Accurate Dependency Parser using Neural Networks*"中的特征集。负责提取这些特征的函数已经在 utils/parser_utils.py 中为你实现好了。这个特征向量由一个单词列表组成（例如栈中的最后一个词，缓冲区的第一个词，从栈的倒数第二个单词到最后一个单词的依赖关系（如果存在的话））。它们能被表示成一个整数列表 $\boldsymbol{w} = [w_1, w_2, \ldots, w_m]$，其中 m 是特征的数量，w_i 是单词在词汇表中的索引，$0 \leq w_i \leq |V|$（$|V|$ 是词汇表的大小）。随后我们的网络为每个单词查询词嵌入，并将词嵌入拼接成单一的输入向量：$\boldsymbol{x} = [\boldsymbol{E}_{w_1}, \ldots, \boldsymbol{E}_{w_m}] \in R^{dm}$。$\boldsymbol{E}$ 是总嵌入矩阵，每个行向量表示一个特定的词。

预测的计算过程如下：

$$\mathbf{h} = \text{ReLU}(\mathbf{x}\mathbf{W} + \mathbf{b}_1)$$
$$\mathbf{l} = \mathbf{h}\mathbf{U} + \mathbf{b}_2$$
$$\hat{\mathbf{y}} = \text{softmax}(l)$$

训练模型以最小化交叉熵损失：

$$J(\theta) = CE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{3} y_i \log \hat{y}_i$$

训练集的损失由所有训练样本的损失求平均后得到。

使用 UAS 分数作为评价指标。UAS 即未标注关联分数，由预测正确的依赖数比上总依赖数得到（模型不预测关系）。

在 parser_model.py 中，你将找到基于 PyTorch 实现这个简单神经网络的主干代码。请完成__init__，embedding_lookup 和 forward 函数实现模型。然后在 run.py 文件中完成 train_for_epoch 和 train 函数。最后执行 python run.py 训练模型，并计算 Penn Treebank（广义依赖标注）测试数据上的预测。

注意事项：

1. 本次作业需要实现线性层和嵌入层。请不要在代码中使用 torch.nn.Linear 或者 torch.nn.Embedding 模块，否则扣分
2. 遵循 TODO 中的命名要求（如有）

提示：

1. self.embed_to_hidden_weight,self.embed_to_hidden_bias,self.hidden_to_logits_weight,self.hidden_to_logits_bias 分别对应（$\boldsymbol{W}$, $\boldsymbol{b}_1$, $\boldsymbol{U}$, $\boldsymbol{b}_2$）
2. 在算法中反向计算，追踪矩阵/向量大小或许有帮助
3. 一旦你实现了 embedding_lookup (e) 或 forward (f)，你可以执行 python parser_model.py -e/-f/-e -f 对函数做可行性测试。通过可行性测试不意味着代码没有 bug
4. debug 的时候，可以执行 python run.py -d，使代码运行在一个小数据集上，这样训练时间不会很长。结束 debug 后，记得移除-d 符号
5. 当在 debug 模式下运行时，你应该在 dev 数据集上得到低于 0.2 的损失值和大

于 65 的 UAS

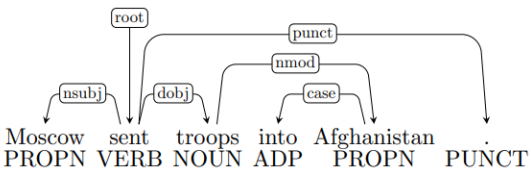
6. 在整个训练集上训练预计花费 1h 左右的时间（debug 模式关闭）
7. debug 模式关闭下，你应该在训练集上得到低于 0.08 的损失，在 dev 数据集上得到高于 87 的 UAS。为了比较，在原始神经依赖解析论文中的模型得到了 92.5 的 UAS，乐意的话，你可以微调模型的超参数（隐层大小，Adam 超参数，训练轮次，等等）以改善性能。

提交物：

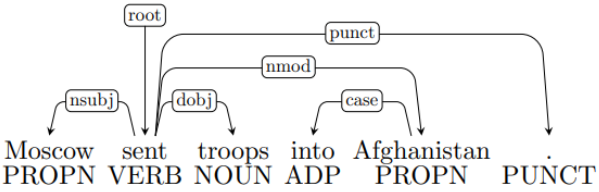
1. parser_transitions.py 中的神经依赖解析器运行逻辑的实现
2. parser_transitions.py 中小批量依赖解析的实现
3. parser_model.py 中神经依赖解析器的实现
4. run.py 中训练函数的实现
5. 记录模型在 dev 数据集上的最佳 UAS，以及在测试集上的最佳 UAS，体现在报告当中

```
Epoch 9 out of 10
100%|██████████████████████████████████████████████████████████| 1848/1848 [00:59<00:00, 31.18it/s]
Average Train Loss: 0.601798660633884
Evaluating on dev set
================================================================================
TESTING
================================================================================
Restoring the best model weights found on the dev set
Final evaluation on test set
2919736it [00:00, 58959305.10it/s]
- test UAS: 88.29
Done!
```

(f) 我们将分析一个依赖解析的例子，理解我们的解析器可能在哪里出错。比如，在这个句子中：



词组 *into Afghanistan* 的依赖关系是错误的，因为词组应该修饰 *sent*，而不是 *troops*，以下是正确的解析结果：



一般而言，存在以下四种解析错误：

1. 介词词组关联错误：

在上述例子中，词组 *into Afghanistan* 是一个介词词组。介词词组关联错误由介词词组被关联到错误的头部词引起（*troops* 是错误的头部词而 *sent* 是正确的头部词）。介词词组的例子还有 *with a rock, before midnight* 和 *under the carpet*。

2. 动词词组关联错误：

在句子 *Leaving the store unattended, I went outside to watch the parade* 中，词组 *leaving the store unattended* 是一个动词词组。动词词组关联错误由动词词组关联到错误的头部词引起（本例中，正确的头部词是 *went*）。

3. 修饰关联错误：

在句子 *I am extremely short* 中，副词 *extremely* 修饰形容词 *short*。修饰关联错误由修饰词关联到错误的头部词引起（本例中，正确的头部词是 *short*）。

4. 协调关联错误：

在句子 *Would you like brown rice or garlic naan?*中，词组 *brown rice* 和 *garlic naan* 都是连接对象，单词 *or* 是协调连词。第二个连接对象（这里对应 *garlic naan*）应该关联到第一个连接对象（这里对应 *brown rice*）。协调关联错误由第二个连接对象关联到错误的头部词引起（本例中，正确的头部词是 rice）。其他的协调连词包括 *and, but* 和 *so*。

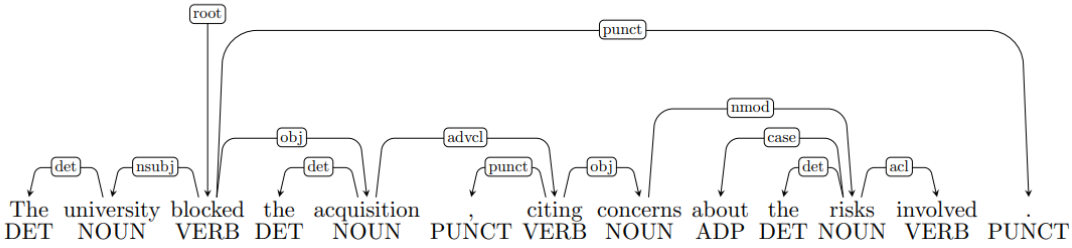
这个问题中有四个从解析器处获得的带有依赖解析的句子。每个句子对应上述四种错误类型中的一种。对每个句子，请指出错误类型、不正确的依赖关系和正确的依赖关系。尽管每个句子只对应一种错误类型，对部分句子而言可能存在多个正确的依赖关系。演示一下：对上面的例子，你应该写出：

- 错误类型：介词词组关联错误

- 错误依赖：troops → Afghanistan
- 正确依赖：sent → Afghanistan

注意：依赖标注存在许多细节和惯例。详情请参考 http://universaldependencies.org 或者简短的导论 PPT http://people.cs.georgetown.edu/nschneid/p/UD-for-English.pdf。你并不需要知道所有这些细节来解答这个问题。在每一个样例中，我们想问的是短语的关联方式，只要看到它们是否修饰正确的头部词即可。特别地，不需要查看依赖边上的标签，只需查看边本身即可。(12 points)

i.

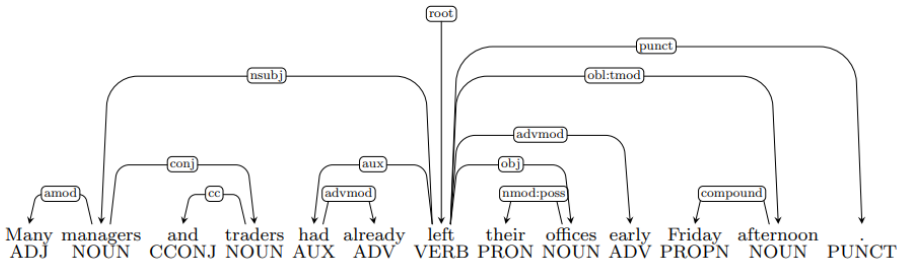


Error type: Verb Phrase Attachment Error

Incorrect dependency: acquisition → citing

Correct dependency: blocked → citing

ii.

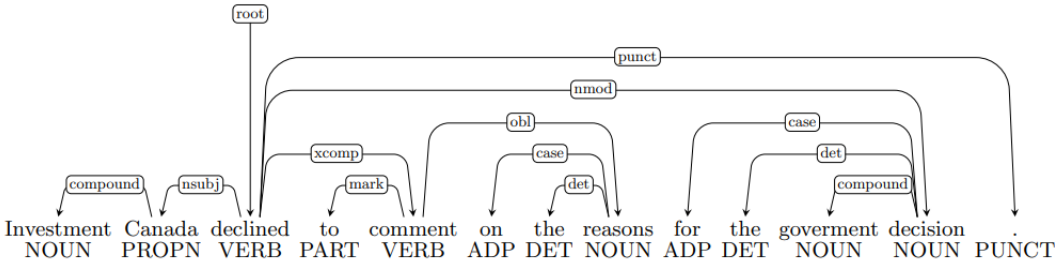


Error type: Modifier Attachment Error

Incorrect dependency: left → early

Correct dependency: afternoon → early

iii.

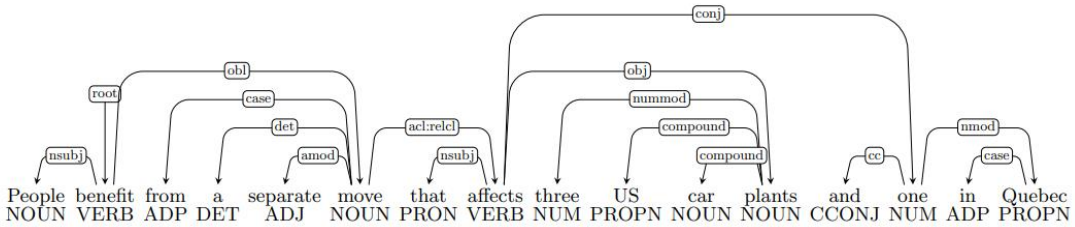


Error type: Prepositional Phrase Attachment Error

Incorrect dependency: declined → decision

Correct dependency: reasons → decision

iv.



Error type: Coordination Attachment Error

Incorrect dependency: affects → one

Correct dependency: plants → one

(g) 在(e)中，解析器使用的特征包括单词和它们的词性标签。请解释使用词性标签作为解析器特征的益处。

My answer:

词性标签能够帮助机器更好地理解句子的结构，减少单词的歧义，对依赖关系产生更多有效约束（例如副词修饰动词或形容词）