CS224n-2019作业笔记 - Science is interesting.

笔记本:	cs224n-2019
创建时间:	2019/11/11 15:01
URL:	https://looperxx.github.io/CS224n-2019-Assignment/

Assignment 03

1. Machine Learning & Neural Networks

(a) Adam Optimizer

回忆一下标准随机梯度下降的更新规则

$$\theta \leftarrow \theta - \alpha \nabla_\theta J_{\text{minibatch}}(\theta)$$

其中，θ 是包含模型所有参数的向量，J 是损失函数，$\nabla_\theta J_{\text{minibatch}}(\theta)$ 是关于minibatch数据上参数的损失函数的梯度，α 是学习率。Adam Optimization使用了一个更复杂的更新规则，并附加了两个步骤。

> **Question 1.a.i**
>
> 首先，Adam使用了一个叫做 momentum **动量**的技巧来跟踪梯度的移动平均值 m
>
> $$m \leftarrow \beta_1 m + (1 - \beta_1) \nabla_\theta J_{\text{minibatch}}(\theta)$$
> $$\theta \leftarrow \theta - \alpha m$$
>
> 其中，β_1 是一个 0 和 1 之间的超参数(通常被设为0.9)。简要说明(不需要用数学方法证明，只需要直观地说明)如何使用m来阻止更新发生大的变化，以及总体上为什么这种小变化可能有助于学习。

Answer 1.a.i :

- 由于超参数 β_1 一般被设为0.9，此时对于移动平均的梯度值 m 而言，主要受到的是之前梯度的移动平均值的影响，而本次计算得到

的梯度将会被缩放为原来的 $1-\beta_1$ 倍，即时本次计算得到的梯度很大（梯度爆炸），这一影响也会被减轻，从而阻止更新发生大的变化。

- 通过减小梯度的变化程度，使得每次的梯度更新更加稳定，从而使模型学习更加稳定，收敛速度更快，并且这也减慢了对于较大梯度值的参数的更新速度，保证其更新的稳定性。

> **Question 1.a.ii**
>
> Adam还通过跟踪梯度平方的移动平均值 v 来使用自适应学习率
>
> $$m \leftarrow \beta_1 m + (1-\beta_1)\nabla_\theta J_{\text{minibatch}}(\theta)$$
> $$v \leftarrow \beta_2 v + (1-\beta_2)(\nabla_\theta J_{\text{minibatch}}(\theta) \odot \nabla_\theta J_{\text{minibatch}}(\theta))$$
> $$\theta \leftarrow \theta - \alpha \odot m/\sqrt{v}$$
>
> 其中，$\odot, /$ 分别表示逐元素的乘法和除法（所以 $z \odot z$ 是逐元素的平方），β_2 是一个 0 和 1 之间的超参数(通常被设为0.99)。因为Adam将更新除以 $\sqrt{v}$，那么哪个模型参数会得到更大的更新？为什么这对学习有帮助？

Answer 1.a.ii :

- 移动平均梯度最小的模型参数将得到较大的更新。
- 一方面，将梯度较小的参数的更新变大，帮助其走出局部最优点（鞍点）；另一方面，将梯度较大的参数的更新变小，使其更新更加稳定。结合以上两个方面，使学习更加快速的同时也更加稳定。

(b) Dropout

Dropout 是一种正则化技术。在训练期间，Dropout 以 p_{drop} 的概率随机设置隐藏层 h 中的神经元为零(每个minibatch中 dropout 不同的神经元),然后将 h 乘以一个常数 γ 。我们可以写为

$$h_{drop} = \gamma d \circ h$$

其中，$d \in \{0,1\}^{D_h}$（D_h 是 h 的大小)是一个掩码向量，其中每个条目都是以 p_{drop} 的概率为 0，以 $1-p_{drop}$ 的概率为 1。γ 是使得 h_{drop} 的期望值为 h 的值

$$E_{p_{drop}}[h_{drop}]_i = h_i, \text{ for all } i \in \{1, \dots, D_h\}$$

Question 1.b.i

γ 必须等于什么(用 p_{drop} 表示)？ 简单证明你的答案。

Answer 1.b.i :

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

证明如下:

$$\sum_i (1 - p_{drop}) h_i = (1 - p_{drop}) E[h]$$

$$\sum_i [h_{drop}]_i = \gamma \sum_i (1 - p_{drop}) h_i = \gamma (1 - p_{drop}) E[h] = E[h]$$

Question 1.b.ii

为什么我们应该只在训练时使用 dropout 而在评估时不使用?

Answer 1.b.ii :

如果我们在评估期间应用 dropout ，那么评估结果将会具有随机性，并不能体现模型的真实性能，违背了正则化的初衷。通过在评估期间禁用 dropout，从而观察模型的性能与正则化的效果，保证模型的参数得到正确的更新。

2. Neural Transition-Based Dependency Parsing

在本节中，您将实现一个基于神经网络的依赖解析器，其目标是在 UAS(未标记依存评分)指标上最大化性能。

依存解析器分析句子的语法结构，在 head words 和 修饰 head words 的单词之间建立关系。你的实现将是一个基于转换的解析器，它逐步构建一个解析。每一步都维护一个局部解析，表示如下

- 一个存储正在被处理的单词的 栈
- 一个存储尚未处理的单词的 缓存
- 一个解析器预测的 依赖 的列表

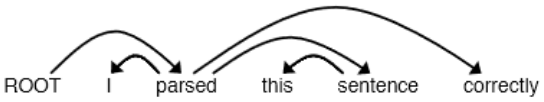
最初,栈只包含 ROOT ，依赖项列表是空的，而缓存则包含了这个句子的所有单词。在每一个步骤中,解析器将对部分解析使用一个转换,直到它的魂村是空的，并且栈大小为1。可以使用以下转换：

- SHIFT：将buffer中的第一个词移出并放到stack上。
- LEFT-ARC：将第二个(最近添加的第二)项标记为栈顶元素的依赖，并从堆栈中删除第二项
- RIGHT-ARC：将第一个(最近添加的第一)项标记为栈中第二项的依赖，并从堆栈中删除第一项

在每个步骤中，解析器将使用一个神经网络分类器在三个转换中决定。

Question 2.a

求解解析句子 “I parsed this sentence correctly” 所需的转换顺序。这句话的依赖树如下所示。在每一步中，给出 stack 和 buffer 的结构，以及本步骤应用了什么转换，并添加新的依赖(如果有的话)。下面提供了以下三个步骤。



Stack	Buffer	New dependency	Transition
[ROOT]	[I, parsed, this, sentence, correctly]		Initial Configuration
[ROOT, I]	[parsed, this, sentence, correctly]		SHIFT
[ROOT, I, parsed]	[this, sentence, correctly]		SHIFT
[ROOT, parsed]	[this, sentence, correctly]	parsed→I	LEFT−ARC

Answer 2.a :

Stack	Buffer	New dependency	Transition
[ROOT]	[I, parsed, this, sentence, correctly]		Initial Configuration
[ROOT, I]	[parsed, this, sentence, correctly]		SHIFT
[ROOT, I, parsed]	[this, sentence, correctly]		SHIFT
[ROOT, parsed]	[this, sentence, correctly]	parsed → I	LEFT-ARC
[ROOT, parsed, this]	[sentence, correctly]		SHIFT
[ROOT, parsed, this, sentence]	[correctly]		SHIFT
[ROOT, parsed, sentence]	[correctly]	sentence → this	LEFT-ARC
[ROOT, parsed]	[correctly]	parsed → sentence	RIGHT-ARC
[ROOT, parsed, correctly]	[]		SHIFT
[ROOT, parsed]	[]	parsed → correctly	RIGHT-ARC
[ROOT]	[]	ROOT → parsed	RIGHT-ARC

Question 2.b

一个包含 n 个单词的句子需要多少步(用 n 表示)才能被解析？简要解释为什么。

Answer 2.b :

包含n个单词的句子需要 $2 \times n$ 步才能完成解析。因为需要进行 n 步的 SHIFT 操作和 共计$n 步的 LEFT-ARC 或 RIGHT-ARC 操作，才能完成解析。（每个单词都需要一次SHIFT和ARC的操作，初始化步骤不计算在内）

Question 2.c

实现解析器将使用的转换机制

Question 2.d

我们的网络将预测哪些转换应该应用于部分解析。我们可以使用它来解析一个句子，通过应用预测出的转换操作，直到解析完成。然而，在对大量数据进行预测时，神经网络的运行速度要高得多(即同时预测了对任何不同部分解析的下一个转换)。我们可以用下面的算法来解析小批次的句子

Algorithm 1 Minibatch Dependency Parsing

Input: `sentences`, a list of sentences to be parsed and `model`, our model that makes parse decisions

Initialize `partial_parses` as a list of PartialParses, one for each sentence in `sentences`
Initialize `unfinished_parses` as a shallow copy of `partial_parses`
while `unfinished_parses` is not empty **do**
    Take the first `batch_size` parses in `unfinished_parses` as a minibatch
    Use the `model` to predict the next transition for each partial parse in the minibatch
    Perform a parse step on each partial parse in the minibatch with its predicted transition
    Remove the completed (empty buffer and stack of size 1) parses from `unfinished_parses`
end while

Return: The `dependencies` for each (now completed) parse in `partial_parses`.

实现minibatch的解析器

我们现在将训练一个神经网络来预测，考虑到栈、缓存和依赖项集合的状态，下一步应该应用哪个转换。首先，模型提取了一个表示当前状态的特征向量。我们将使用原神经依赖解析论文中的特征集合：A Fast and Accurate Dependency Parser using Neural Networks。这个特征向量由标记列表(例如在栈中的最后一个词，缓存中的第一个词，栈中第二到最后一个字的依赖(如果有))组成。它们可以被表示为整数的列表 $[w_1, w_2, \dots, w_m]$，m是特征的数量，每个 $0 \le w_i < |V|$ 是词汇表中的一个token的索引($|V|$是词汇量)。首先，我们的网络查找每个单词的嵌入，并将它们连接成一个输入向量：

$$x = [E_{w_1}, \dots, E_{w_m}] \in R^{dm}$$

其中 $E \in R^{|V| \times d}$ 是嵌入矩阵，每一行 E_w 是一个特定的单词 w 的向量。接着我们可以计算我们的预测：

$$h = \mathrm{ReLU}(xW + b_1)$$
$$l = \mathrm{ReLU}(hU + b_2)$$
$$\hat{y} = \mathrm{softmax}(l)$$

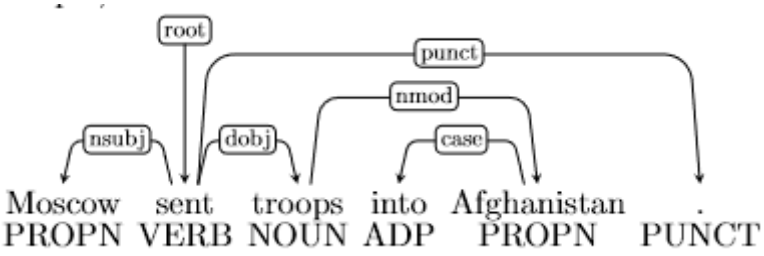
其中， h 指的是隐藏层，l 是其分数，$\hat{y}$ 指的是预测结果，$\text{ReLU}(z) = \max(z, 0)$ 。我们使用最小化交叉熵损失来训练模型

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

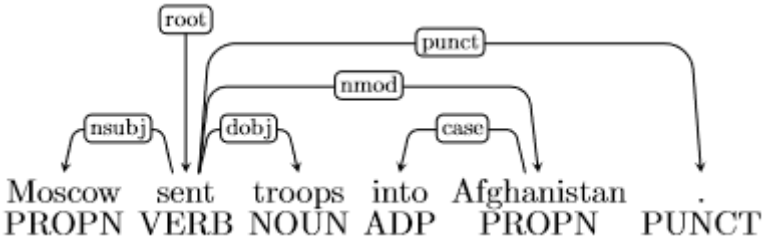
训练集的损失为所有训练样本的 $J(\theta)$ 的平均值。

Question 2.f

我们想看看依赖关系解析的例子，并了解像我们这样的解析器在什么地方可能是错误的。例如，在这个句子中:



依赖 into Afghanistan 是错的，因为这个短语应该修饰 sent (例如 sent into Afghanistan) 而不是 troops (因为 troops into Afghanistan 没有意义)。下面是正确的解析:



一般来说，以下是四种解析错误:

- **Prepositional Phrase Attachment Error** 介词短语连接错误：在上面的例子中，词组 into Afghanistan 是一个介词短语。介词短语连接错误是指介词短语连接到错误的 head word 上(在本例中，troops 是错误的 head word ， sent 是正确的 head word)。介词短语的更多例子包括with a rock, before midnight和under the carpet。

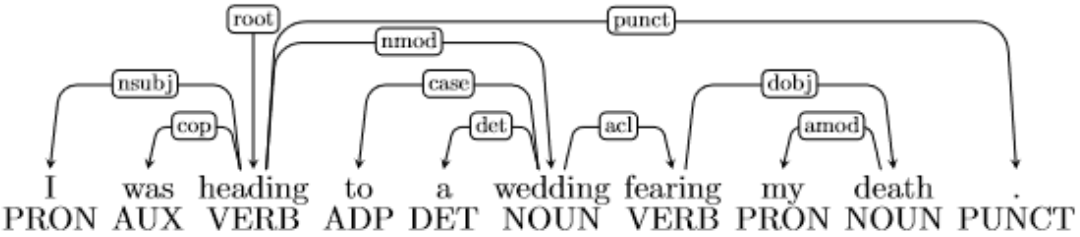
- **Verb Phrase Attachment Error** 动词短语连接错误：在句子 leave the store alone, I went out to watch the parade 中，短语 leave the store alone 是动词短语。动词短语连接错误是指一个动词短语连接到错误的 head word 上(在本例中，正确的头词是 went)。
- **Modifier Attachment Error** 修饰语连接错误：在句子 I am extremely short 中，副词extremely 是形容词 short 的修饰语。修饰语附加错误是修饰语附加到错误的 head word 上时发生的错误(在本例中，正确的头词是 short)。
- **Coordination Attachment Error** 协调连接错误：在句子 Would you like brown rice or garlic naan? 中， brown rice 和garlic naan都是连词，or是并列连词。第二个连接词(这里是 garlic naan)应该连接到第一个连接词(这里是brown rice)。协调连接错误是当第二个连接词附加到错误的 head word 上时(在本例中，正确的头词是rice)。其他并列连词包括and, but和so。

在这个问题中有四个句子，其中包含从解析器获得的依赖项解析。每个句子都有一个错误，上面四种类型都有一个例子。对于每个句子，请说明错误的类型、不正确的依赖项和正确的依赖项。为了演示:对于上面的例子，您可以这样写：

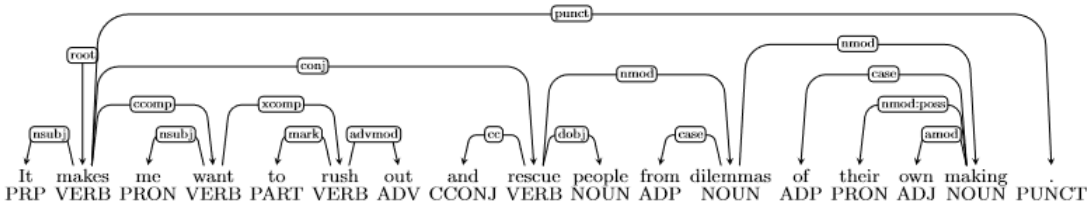
- Error type: Prepositional Phrase Attachment Error
- Incorrect dependency: troops → Afghanistan
- Correct dependency: sent → Afghanistan

注意：依赖项注释有很多细节和约定。如果你想了解更多关于他们的信息，你可以浏览UD网站:http://universaldependencies.org。然而，你不需要知道所有这些细节就能回答这个问题。在每一种情况下，我们都在询问短语的连接，应该足以看出它们是否修饰了正确的head。特别是，你不需要查看依赖项边缘上的标签——只需查看边缘本身就足够了。

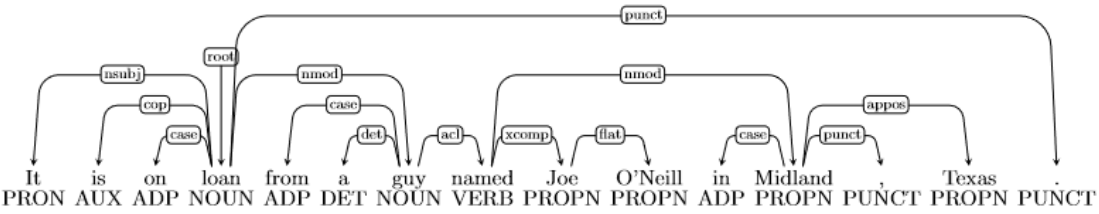
Answer 2.f



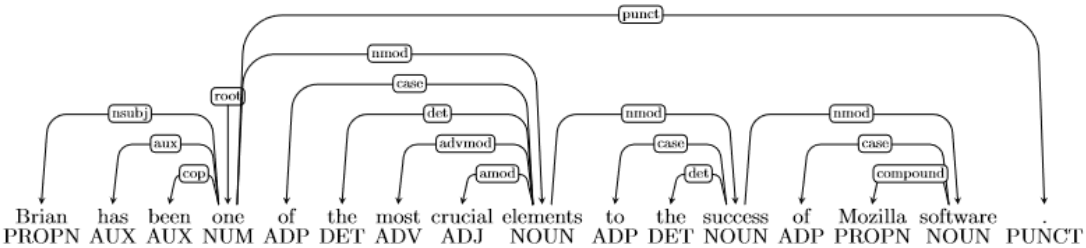
- Error type: Verb Phrase Attachment Error
- Incorrect dependency: wedding → fearing
- Correct dependency: heading → fearing



- Error type: Coordination Attachment Error
- Incorrect dependency: makes → rescue
- Correct dependency: rush → rescue



- Error type: Prepositional Phrase Attachment Error
- Incorrect dependency: named → Midland
- Correct dependency: guy → Midland



- Error type: Modifier Attachment Error

- Incorrect dependency: elements → most
- Correct dependency: crucial → most