

CS224n-2019 Assignment

本文档将记录作业中的要点以及问题的答案

课程笔记参见我的[博客](#)，并在博客的[Repo](#)中提供笔记源文件的下载

Assignment 01

- 逐步完成共现矩阵的搭建，并调用 `sklearn.decomposition` 中的 `TruncatedSVD` 完成传统的基于SVD的降维算法
- 可视化展示，观察并分析其在二维空间下的聚集情况。
- 载入Word2Vec，将其与SVD得到的单词分布情况进行对比，分析两者词向量的不同之处。
- 学习使用 `gensim`，使用 `Cosine Similarity` 分析单词的相似度，对比单词和其同义词与反义词的 `Cosine Distance`，并尝试找到正确的与错误的类比样例
- 探寻Word2Vec向量中存在的 `Independent Bias` 问题

Assignment 02

1 Written: Understanding word2vec

$$P(O = o | C = c) = \frac{\exp(\mathbf{u}_o^T \mathbf{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\mathbf{u}_w^T \mathbf{v}_c)}$$

$$J_{\text{naive-softmax}}(\mathbf{v}_c, o, U) = -\log P(O = o | C = c)$$

真实(离散)概率分布 p 与另一个分布 q 的交叉熵损失为 $-\sum_i p_i \log(q_i)$

Question a

Show that the naive-softmax loss given in Equation (2) is the same as the cross-entropy loss between y and \hat{y} ; i.e., show that

$$-\sum_{w \in \text{Vocab}} y_w \log(\hat{y}_w) = -\log(\hat{y}_o)$$

Your answer should be one line.

Answer a :

因为 y 是独热向量，所以 $-\sum_{w \in \text{Vocab}} y_w \log(\hat{y}_w) = -y_o \log(\hat{y}_o) - \sum_{w \in \text{Vocab}, w \neq o} y_w \log(\hat{y}_w) = -\log(\hat{y}_o)$

Question b

Compute the partial derivative of $J_{\text{naive-softmax}}(\mathbf{v}_c, o, U)$ with respect to \mathbf{v}_c . Please write your answer in terms of y, \hat{y}, U .

Answer b :

$$\begin{aligned} \frac{\partial J(\mathbf{v}_c, o, U)}{\partial \mathbf{v}_c} &= -\frac{\partial(\mathbf{u}_o^T \mathbf{v}_c)}{\partial \mathbf{v}_c} + \frac{\partial(\log(\sum_w \exp(\mathbf{u}_w^T \mathbf{v}_c)))}{\partial \mathbf{v}_c} \\ &= -\mathbf{u}_o + \frac{1}{\sum_w \exp(\mathbf{u}_w^T \mathbf{v}_c)} \frac{\partial(\sum_w \exp(\mathbf{u}_w^T \mathbf{v}_c))}{\partial \mathbf{v}_c} \\ &= -\mathbf{u}_o + \sum_w \frac{\exp(\mathbf{u}_w^T \mathbf{v}_c) \mathbf{u}_w}{\sum_w \exp(\mathbf{u}_w^T \mathbf{v}_c)} \\ &= -\mathbf{u}_o + \sum_w p(O = w | C = c) \mathbf{u}_w \\ &\quad \wedge \\ &= -\mathbf{u}_o + \sum_w y_w \mathbf{u}_w \\ &= U(\hat{y} - y) \end{aligned}$$

Question c

Compute the partial derivatives of $J_{\text{naive-softmax}}(\mathbf{v}_c, o, U)$ with respect to each of the 'outside' word vectors, \mathbf{u}_w 's. There will be two cases: when $w = o$, the true 'outside' word vector, and $w \neq o$, for all other words. Please write your answer in terms of y, \hat{y}, U .

Answer c :

$$\frac{\partial J(\mathbf{v}_c, o, U)}{\partial \mathbf{u}_o} \quad \frac{\partial(\mathbf{u}_o^T \mathbf{v}_c)}{\partial \mathbf{u}_o} \quad \frac{\partial(\log(\sum_w \exp(\mathbf{u}_w^T \mathbf{v}_c)))}{\partial \mathbf{u}_o}$$



目录

Assignment 01

Assignment 02

1 Written: Understanding word2vec

2 Coding: Implementing word2vec

Assignment 03

1. Machine Learning & Neural Networks

2. Neural Transition-Based Dependency Parsing

Assignment 04

1. Neural Machine Translation with RNNs

2. Analyzing NMT Systems

Reference

$$\frac{\partial}{\partial u_w} = - \frac{\partial}{\partial u_w} + \frac{\partial}{\partial u_w}$$

When $w \neq o$:

$$\begin{aligned} \frac{\partial J(v_c, o, U)}{\partial u_w} &= 0 + p(O = w | C = c) v_c \\ &= \hat{y}_w v_c \end{aligned}$$

When $w = o$:

$$\begin{aligned} \frac{\partial J(v_c, o, U)}{\partial u_w} &= -v_c + p(O = o | C = c) v_c \\ &= \hat{y}_w v_c - v_c \\ &= (\hat{y}_w - 1) v_c \end{aligned}$$

Then :

$$\frac{\partial J(v_c, o, U)}{\partial U} = v_c (\hat{y} - y)^T$$

? Question d

The sigmoid function is given by the follow Equation :

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

Please compute the derivative of $\sigma(x)$ with respect to x , where x is a vector.

Answer d :

$$\begin{aligned} \frac{\partial \sigma(x_i)}{\partial x_i} &= \frac{1}{(1 + \exp(-x_i))^2} \exp(-x_i) = \sigma(x_i)(1 - \sigma(x_i)) \\ \frac{\partial \sigma(x)}{\partial x} &= \left[\frac{\partial \sigma(x_i)}{\partial x_i} \right]_{d \times d} \\ &= \begin{bmatrix} \sigma'(x_1) & 0 & \dots & 0 \\ 0 & \sigma'(x_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma'(x_d) \end{bmatrix} \\ &= \text{diag}(\sigma'(x)) \end{aligned}$$

? Question e

Now we shall consider the Negative Sampling loss, which is an alternative to the Naive Softmax loss. Assume that K negative samples (words) are drawn from the vocabulary. For simplicity of notation we shall refer to them as w_1, w_2, \dots, w_K and their outside vectors as u_1, \dots, u_K . Note that $o \notin \{w_1, \dots, w_K\}$. For a center word c and an outside word o , the negative sampling loss function is given by:

$$J_{\text{neg-sample}}(v_c, o, U) = -\log(\sigma(u_o^\top v_c)) - \sum_{k=1}^K \log(\sigma(-u_k^\top v_c))$$

for a sample w_1, w_2, \dots, w_K , where $\sigma(\cdot)$ is the sigmoid function.

Please repeat parts b and c, computing the partial derivatives of $J_{\text{neg-sample}}$ respect to v_c , with respect to u_o , and with respect to a negative sample u_k . Please write your answers in terms of the vectors u_o , v_c , and u_k , where $k \in [1, K]$. After you've done this, describe with one sentence why this loss function is much more efficient to compute than the naive-softmax loss. Note, you should be able to use your solution to part (d) to help compute the necessary gradients here.

Answer e :

For v_c :

$$\begin{aligned} \frac{\partial J_{\text{neg-sample}}}{\partial v_c} &= (\sigma(u_o^\top v_c) - 1) u_o + \sum_{k=1}^K (1 - \sigma(-u_k^\top v_c)) u_k \\ &= (\sigma(u_o^\top v_c) - 1) u_o + \sum_{k=1}^K \sigma(u_k^\top v_c) u_k \end{aligned}$$

For u_o , Remeber : $o \notin \{w_1, \dots, w_K\}$ 🤖 :

$$\frac{\partial J_{\text{neg-sample}}}{\partial u_o} = (\sigma(u_o^\top v_c) - 1) v_c$$

For u_k :

$$\frac{\partial J}{\partial u_k} = -(\sigma(-u_k^\top v_c) - 1) v_c = \sigma(u_k^\top v_c) v_c, \text{ for } k = 1, 2, \dots, K$$

For naive softmax loss function:

$$\frac{\partial J(v_c, o, U)}{\partial v_c} = U(\hat{y} - y)$$
$$\frac{\partial J(v_c, o, U)}{\partial U} = v_c(\hat{y} - y)^T$$

For negative sampling loss function:

$$\frac{\partial J}{\partial v_c} = \left(\sigma(u_o^\top v_c) - 1 \right) u_o + \sum_{k=1}^K \sigma(u_k^\top v_c) u_k = \sigma(-u_o^\top v_c) u_o + \sum_{k=1}^K \sigma(u_k^\top v_c) u_k$$
$$\frac{\partial J}{\partial u_o} = \left(\sigma(u_o^\top v_c) - 1 \right) v_c = \sigma(-u_o^\top v_c) v_c$$
$$\frac{\partial J}{\partial u_k} = \sigma(u_k^\top v_c) v_{c^*} \quad \text{for all } k = 1, 2, \dots, K$$

从求得的偏导数中我们可以看出，原始的softmax函数每次对 v_c 进行反向传播时，需要与 output vector matrix 进行大量且复杂的矩阵运算，而负采样中的计算复杂度则不再与词表大小 V 有关，而是与采样数量 K 有关。

? Question f

Suppose the center word is $c = w_t$ and the context window is $[w_{t-m}, \dots, w_{t-1}, w_t, w_{t+1}, \dots, w_{t+m}]$, where m is the context window size. Recall that for the skip-gram version of **word2vec**, the total loss for the context window is

$$J_{\text{skip-gram}}(v_c, w_{t-m}, \dots, w_{t+m}, U) = \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} J(v_c, w_{t+j}, U)$$

Here, $J(v_c, w_{t+j}, U)$ represents an arbitrary loss term for the center word $c = w_t$ and outside word w_{t+j} . $J(v_c, w_{t+j}, U)$ could be $J_{\text{naive-softmax}}(v_c, w_{t+j}, U)$ or $J_{\text{neg-sample}}(v_c, w_{t+j}, U)$, depending on your implementation.

Write down three partial derivatives:

$$\frac{\partial J_{\text{skip-gram}}(v_c, w_{t-m}, \dots, w_{t+m}, U)}{\partial U} \frac{\partial U}{\partial v_c} \frac{\partial J_{\text{skip-gram}}(v_c, w_{t-m}, \dots, w_{t+m}, U)}{\partial v_c} \frac{\partial J_{\text{skip-gram}}(v_c, w_{t-m}, \dots, w_{t+m}, U)}{\partial v_w} \text{ when } w \neq c$$

Write your answers in terms of $\frac{\partial J(v_c, w_{t+j}, U)}{\partial U}$ and $\frac{\partial J(v_c, w_{t+j}, U)}{\partial v_c}$. This is very simple - each solution should be one line.

Once you're done: Given that you computed the derivatives of $\frac{\partial J(v_c, w_{t+j}, U)}{\partial U}$ with respect to all the model parameters U and V in parts a to c, you have now computed the derivatives of the full loss function $J_{\text{skip-gram}}$ with respect to all parameters. You're ready to implement **word2vec**!

Answer f :

$$\frac{\partial J_{\text{sg}}}{\partial U} = \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial J(v_c, w_{t+j}, U)}{\partial U} \frac{\partial J_{\text{sg}}}{\partial v_c} = \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial J(v_c, w_{t+j}, U)}{\partial v_c} \frac{\partial J_{\text{sg}}}{\partial v_w} = 0 \text{ (when } w \neq c \text{)}$$

2 Coding: Implementing word2vec

word2vec.py

本部分要求实现 `sigmoid`, `naiveSoftmaxLossAndGradient`, `negSamplingLossAndGradient`, `skipgram` 四个函数，主要考察对第一部分中反向传播计算结果的实现。代码实现中，通过优化偏导数结合偏导数计算结果与 $\sigma(x) + \sigma(-x) = 1$ 对公式进行转化，从而实现了全矢量化。这部分需要大家自行结合代码与公式进行推导。

sgd.py

实现 SGD

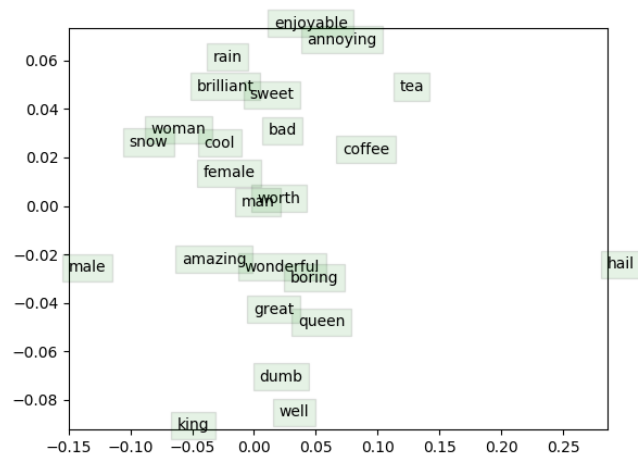
$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} l(\theta)$$

run.py

首先要说明的是，这个真的要跑好久 😓

? Question

Briefly explain in at most three sentences what you see in the plot.



上图是经过训练的词向量的可视化。我们可以注意到一些模式：

- 近义词被组合在一起，比如 amazing 和 wonderful，woman 和 female。
 - 但是 man 和 male 却距离较远
- 反义词可能因为经常属于同一上下文，它们也会与同义词一起出现，比如 enjoyable 和 annoying。
- man:king::woman:queen 以及 queen:king::female:male 形成的两条直线基本平行

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 \mathbf{m}

$$\begin{aligned}\mathbf{m} &\leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \nabla_{\theta} J_{\text{minibatch}}(\theta) \\ \theta &\leftarrow \theta - \alpha \mathbf{m}\end{aligned}$$

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 \mathbf{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 \mathbf{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 \mathbf{v} 来使用自适应学习率

$$\begin{aligned}\mathbf{m} &\leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \nabla_{\theta} J_{\text{minibatch}}(\theta) \\ \mathbf{v} &\leftarrow \beta_2 \mathbf{v} + (1 - \beta_2) \left(\nabla_{\theta} J_{\text{minibatch}}(\theta) \odot \nabla_{\theta} J_{\text{minibatch}}(\theta) \right) \\ \theta &\leftarrow \theta - \alpha \odot \mathbf{m} / \sqrt{\mathbf{v}}\end{aligned}$$

其中， \odot / \oslash 分别表示逐元素的乘法和除法（所以 $\mathbf{z} \oslash \mathbf{z}$ 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 $\sqrt{\mathbf{v}}$ ，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值（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

$$\mathbf{h}_{drop} = \gamma \mathbf{d} \circ \mathbf{h}$$

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

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

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

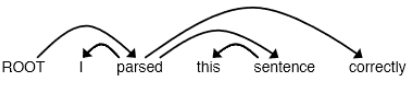
最初,栈只包含 ROOT，依赖项列表是空的，而缓存则包含了这个句子的所有单词。在每一个步骤中,解析器将对部分解析使用一个转换,直到它的缓存是空的，并且栈大小为1。可以使用以下转换：

- 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_pares as a list of PartialPares, one for each sentence in sentences
Initialize unfinished_pares as a shallow copy of partial_pares
while unfinished_pares is not empty **do**
 Take the first batch_size parses in unfinished_pares 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_pares
end while

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

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 $0 \leq w_i < |V|$ 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：

$$\mathbf{x} = \left[\mathbf{E}_{w_1}, \dots, \mathbf{E}_{w_m} \right] \in \mathbb{R}^{dm}$$

其中 $\mathbf{E} \in \mathbb{R}^{|V| \times d}$ 是嵌入矩阵，每一行 \mathbf{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：

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

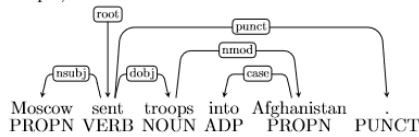
其中， \mathbf{h} 指的是隐藏层， \mathbf{l} 是其分数， $\hat{\mathbf{y}}$ 指的是预测结果， $\text{ReLU}(z) = \max(z, 0)$ 。我们使用最小化交叉熵损失来训练模型

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

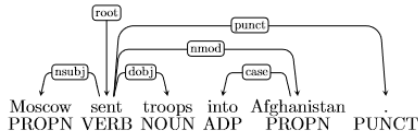
训练集的损失为所有训练样本的 $J(\theta)$ 的平均值。

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：

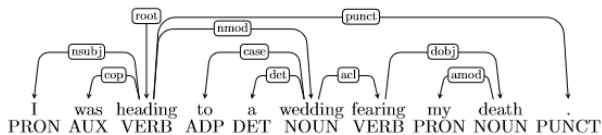
- Prepositional Phrase Attachment Error** 介词短语连接错误：在上面的例子中，词组 into Afghanistan 是一个介词短语。介词短语连接错误是指介词短语连接到错误的 head word 上(在本例中，troops 是错误的 head word，sent 是正确的 head word)。介词短语的更多例子包括 with a rock, before midnight 和 under the carpet。
- Verb Phrase Attachment Error** 动词短语连接错误：在句子 leave the store alone, I went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

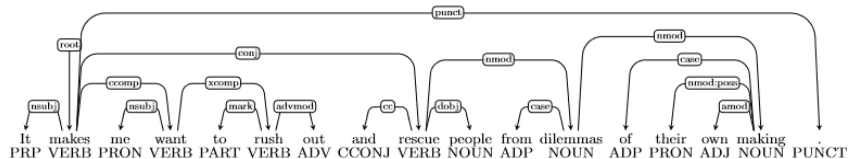
- Error type: Prepositional Phrase Attachment Error
- Incorrect dependency: troops → Afghanistan
- Correct dependency: sent → Afghanistan

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。

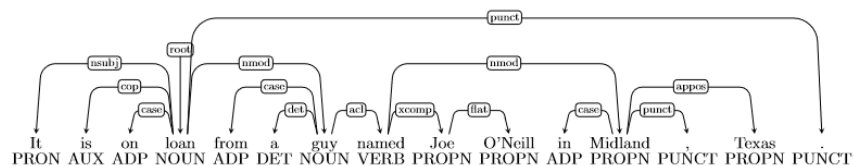
Answer 2.f



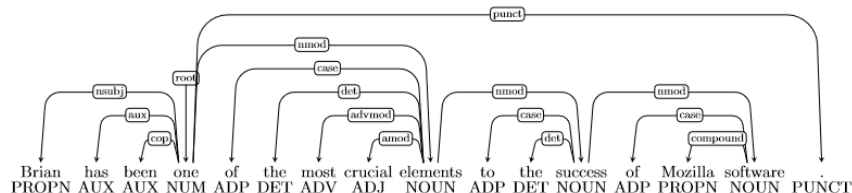
- Error type: Verb Phrase Attachment Error
- Incorrect dependency: wedding → fearing
- Correct dependency: heading → fearing



- 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

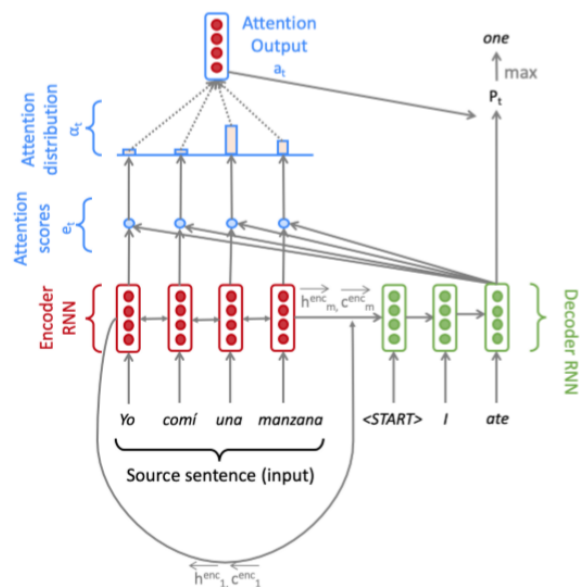


- Error type: Modifier Attachment Error
- Incorrect dependency: elements → most
- Correct dependency: crucial → most

Assignment 04

1. Neural Machine Translation with RNNs

在机器翻译中，我们的目标是将一个句子从源语言(如西班牙语)转换成目标语言(如英语)。在本作业中，我们将注意实现一个序列到序列(Seq2Seq)网络，以建立一个神经机器翻译(NMT)系统。在本节中，我们描述了使用双向LSTM编码器和单向LSTM解码器的NMT系统的训练过程。



上图是使用乘法注意力的Seq2Seq模型，显示了解码器的第三步。注意，为了可读性，我们不描绘前一个组合输出与解码器输入的连接。

给定源语言中的一个句子，我们从词嵌入矩阵中查找单词嵌入，得到 $x_1, \dots, x_m | x_i \in \mathbb{R}^{e \times 1}$ ，其中 m 为源语句的长度， e 为嵌入大小。我们将这些嵌入提供给双向编码器，为正向 (\rightarrow) 和反向 (\leftarrow) LSTM 生成隐藏状态和单元状态。前向和后向的版本连接起来，以得到隐藏状态 $\mathbf{h}_i^{\text{enc}}$ 和单元状态 $\mathbf{c}_i^{\text{enc}}$

$$\mathbf{h}_i^{\text{enc}} = [\mathbf{h}_i^{\text{enc}}, \mathbf{h}_i^{\text{enc}}] \text{ where } \mathbf{h}_i^{\text{enc}} \in \mathbb{R}^{2h \times 1}, \mathbf{h}_i^{\text{enc}}, \mathbf{h}_i^{\text{enc}} \in \mathbb{R}^{h \times 1} \quad 1 \leq i \leq m$$

$$\mathbf{c}_i^{\text{enc}} = [\mathbf{c}_i^{\text{enc}}, \mathbf{c}_i^{\text{enc}}] \text{ where } \mathbf{c}_i^{\text{enc}} \in \mathbb{R}^{2h \times 1}, \mathbf{c}_i^{\text{enc}}, \mathbf{c}_i^{\text{enc}} \in \mathbb{R}^{h \times 1} \quad 1 \leq i \leq m$$

然后，我们使用编码器的最终隐藏状态和最终单元状态的线性投影，初始化解码器的第一个隐藏状态 $\mathbf{h}_0^{\text{dec}}$ 和单元状态 $\mathbf{c}_0^{\text{dec}}$

$$\begin{aligned} \overleftarrow{h_0^{\text{enc}}} &= [\overleftarrow{h_1^{\text{enc}}}, \overrightarrow{h_m^{\text{enc}}}] \text{ where } \mathbf{h}_0^{\text{enc}} \in \mathbb{R}^{h \times 1}, \mathbf{W}_h \in \mathbb{R}^{h \times 2h} \\ \mathbf{c}_0^{\text{enc}} &= [\overleftarrow{c_1^{\text{enc}}}, \overrightarrow{c_m^{\text{enc}}}] \text{ where } \mathbf{c}_0^{\text{enc}} \in \mathbb{R}^{h \times 1}, \mathbf{W}_c \in \mathbb{R}^{h \times 2h} \end{aligned}$$

初始化解码器之后，现在必须用目标语言为它提供匹配的句子。在第 t 步，我们查找第 t 个单词的嵌入， $\mathbf{y}_t \in \mathbb{R}^{e \times 1}$ 。然后，我们将 \mathbf{y}_t 与前一个时间步的 combined-output 组合输出向量 $\mathbf{o}_{t-1} \in \mathbb{R}^{h \times 1}$ 连接起来(我们将在下一页解释这是什么！)，得到 $\mathbf{y}_t \in \mathbb{R}^{(e+h) \times 1}$ 。注意，对于第一个目标单词(即 start 标记)， \mathbf{o}_0 是一个零向量。然后将 \mathbf{y}_t 作为输入输入到解码器 LSTM 中。

$$\mathbf{h}_t^{\text{dec}}, \mathbf{c}_t^{\text{dec}} = \text{Decoder} \left(\overleftarrow{\mathbf{y}_t}, \mathbf{h}_{t-1}^{\text{dec}} \right) \text{ where } \mathbf{h}_t^{\text{dec}} \in \mathbb{R}^{h \times 1}, \mathbf{c}_t^{\text{dec}} \in \mathbb{R}^{h \times 1}$$

然后我们用 $\mathbf{h}_t^{\text{dec}}$ 来计算在 $\mathbf{h}_0^{\text{enc}}, \dots, \mathbf{h}_m^{\text{enc}}$ 上的乘法注意

$$\mathbf{e}_{t,i} = \left(\mathbf{h}_t^{\text{dec}} \right)^T \mathbf{W}_{\text{attProj}} \mathbf{h}_i^{\text{enc}} \text{ where } \mathbf{e}_t \in \mathbb{R}^{m \times 1}, \mathbf{W}_{\text{attProj}} \in \mathbb{R}^{h \times 2h} \quad 1 \leq i \leq m \alpha_t = \text{Softmax} \left(\mathbf{e}_t \right) \text{ where } \alpha_t \in \mathbb{R}^{m \times 1} \mathbf{a}_t = \sum_i^m \alpha_{t,i} \mathbf{h}_i^{\text{enc}} \text{ where } \mathbf{a}_t \in \mathbb{R}^{2h \times 1}$$

现在，我们将注意力输出 α_t 与解码器隐藏状态 $\mathbf{h}_t^{\text{dec}}$ 连接起来，并将其通过线性层 Tanh 和 Dropout 来获得组合输出向量 \mathbf{o}_t 。

$$\begin{aligned} \mathbf{u}_t &= \left[\mathbf{a}_t; \mathbf{h}_t^{\text{dec}} \right] \text{ where } \mathbf{u}_t \in \mathbb{R}^{3h \times 1} \\ \mathbf{v}_t &= \mathbf{W}_u \mathbf{u}_t \text{ where } \mathbf{v}_t \in \mathbb{R}^{h \times 1}, \mathbf{W}_u \in \mathbb{R}^{h \times 3h} \\ \mathbf{o}_t &= \text{Dropout} \left(\text{Tanh} \left(\mathbf{v}_t \right) \right) \text{ where } \mathbf{o}_t \in \mathbb{R}^{h \times 1} \end{aligned}$$


然后，在第 t 个时间步长时，得到目标词的概率分布 \mathbf{P}_t

$$\mathbf{P}_t = \text{Softmax} \left(\mathbf{W}_{\text{vocab}} \mathbf{o}_t \right) \text{ where } \mathbf{P}_t \in \mathbb{R}^{V_t \times 1}, \mathbf{W}_{\text{vocab}} \in \mathbb{R}^{V_t \times h}$$

这里， V_t 是目标词汇表的大小。最后，为了训练网络，我们计算了 $\mathbf{P}_t, \mathbf{g}_t$ 之间的 softmax 交叉熵损失， \mathbf{g}_t 是时间步 t 的目标词的 one-hot 向量

$$J_t(\theta) = CE \left(\mathbf{P}_t, \mathbf{g}_t \right)$$

在这里， θ 代表所有的模型参数， $J_t(\theta)$ 是解码器第 t 步的损失。现在我们已经描述了该模型，让我们尝试将其实现为西班牙语到英语的翻译！



Pytorch Bidirectional RNNs Note

Pytorch 中的 RNNs，返回的 **out** 的 shape 为 (seq_len, batch, num_directions * hidden_size)

- 转换为 (seq_len, batch, num_directions, hidden_size) 后，num_directions 中的顺序是先 forward 再 backward，并且 forward 和 backward 的 hidden state 的顺序是相反的，即 `out[0][0][0]` 是 forward 的第一个时间步的结果，而 `out[0][0][1]` 是 backward 的最后一个时间步的结果。此外，**out** 只包含最后一层的结果

但对于 **h_n** (**c_n** 同理) 而言，shape 为 (num_layers * num_directions, batch, hidden_size)，保存的是 forward 和 backward 的最后一个时间步的结果。


- 转换为 (num_layers, num_directions, batch, hidden_size) 后，第一维的 num_layers 和真实的 layer 层数一一对应，即 `h_n[1][0][0]` 与 `out[-1][0][0]` 相等，`h_n[1][1][0]` 与 `out[0][0][1]`。


Question 1.g

首先解释(大约三句话) masks 对整个注意力计算有什么影响。然后(用一两句话)解释为什么有必要这样使用 masks。

Answer 1.g

- 使用 masks 将句子中的 pad token 的分数赋值为 $-\inf$ ，从而使得 softmax 作用后获得的 attention 分布中，pad token 的 attention 概率值近似为 0
- attention score / distributions 计算的是 decoder 中某一时间步上的 target word 对 encoder 中的所有 source word 的注意力概率，而 pad token 只是用于 mini-batch，并没有任何语言意义，target word 无须为其分散注意力，所以需要 masks 过滤掉 pad token


Question 1.j

在课堂上，我们学习了点积注意、乘法注意和加法注意。请就其他两种注意机制中的任何一种，提供每种注意机制可能的优点和缺点

- 点积注意 $\mathbf{e}_{t,i} = \mathbf{s}_i^T \mathbf{h}_t$
- 乘法注意 $\mathbf{e}_{t,i} = \mathbf{s}_i^T \mathbf{W} \mathbf{h}_t$
- 加法注意 $\mathbf{e}_{t,i} = \mathbf{v}^T (\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}_i)$

Answer 1.j

	优点	缺点
点积注意力	不需要额外的线性映射层	s_r, h_l 必须有同样的维度
乘法注意力	s_r, h_l 不需要有同样的维度并且因为可以使用高效率的矩阵乘法，比加法注意力要更快更省内存	增加了训练参数
加法注意力	高维时的表现更好	训练参数更多（两个参数矩阵以及注意力的维度）

2. Analyzing NMT Systems

Question 2.a

这里，我们展示了在NMT模型的输出中发现的一系列错误(与您刚刚训练的模型相同)。对于西班牙语源句的每个示例，标准英文翻译，以及NMT(即，“模型”)，请你：

- 识别NMT翻译中的错误
- 提供模型可能出错的原因(由于特定的语言构造或特定的模型限制)
- 描述一种可能的方法，我们可以改变NMT系统，以修复观察到的错误

下面是您应该按照上面描述的那样分析的翻译。请注意，标记了下划线的单词是词汇表外的单词

i. (2 points) Source Sentence: *Aquí otro de mis favoritos, “La noche estrellada”.*
Reference Translation: *So another one of my favorites, “The Starry Night”.*
NMT Translation: *Here’s another favorite of my favorites, “The Starry Night”.*

ii. (2 points) Source Sentence: *Ustedes saben que lo que yo hago es escribir para los niños, y, de hecho, probablemente soy el autor para niños, ms ledo en los EEUU.*
Reference Translation: *You know, what I do is write for children, and I’m probably America’s most widely read children’s author, in fact.*
NMT Translation: *You know what I do is write for children, and in fact, I’m probably the author for children, more reading in the U.S.*

iii. (2 points) Source Sentence: *Un amigo me hizo eso – Richard Bolingbroke.*
Reference Translation: *A friend of mine did that – Richard Bolingbroke.*
NMT Translation: *A friend of mine did that – Richard <unk>*

iv. (2 points) Source Sentence: *Solo tienes que dar vuelta a la manzana para verlo como una epifanía.*
Reference Translation: *You’ve just got to go around the block to see it as an epiphany.*
NMT Translation: *You just have to go back to the apple to see it as a epiphany.*

v. (2 points) Source Sentence: *Ella salvó mi vida al permitirme entrar al baño de la sala de profesores.*
Reference Translation: *She saved my life by letting me go to the bathroom in the teachers’ lounge.*
NMT Translation: *She saved my life by letting me go to the bathroom in the women’s room.*

vi. (2 points) Source Sentence: *Eso es más de 100,000 hectáreas.*
Reference Translation: *That’s more than 250 thousand acres.*
NMT Translation: *That’s over 100,000 acres.*

Answer 2.a

Error: “ **favorite** of my favorites”

Reason: 特定的语言构造，低资源语言对

Possible fix: 尝试在这类语言对上添加更多的训练数据

Error: “ **more reading** in the U.S.” 语义错误

Reason: 特定的语言构造，模型对语义的理解不足，需要增大模型的容量以增强理解能力

Possible fix: 增大Hidden_size

Error: "Richard \<unk>"

Reason: 模型限制，Bolingbroke 是词表外的单词

Possible fix: 对此类姓名中出现的词加以处理，比如直接添加到词表中

Processing math: 100%

Error: "go back to the apple "

- Reason: 模型限制, "manzana" 有丰富的含义, 包括 apple 苹果和 block 街区。"block"在西班牙语中的表达方式比 "apple" 在西班牙语中的表达方式更多。然而, 在训练集中, "manzana"更多地表示"apple", 而不是"block"。
- Possible fix: 在训练集中添加更多的关于 "manzana" 表示 "block" 的数据, 保持多重含义的训练不失衡

- Error: "go to the bathroom in the women's room"
- Reason: 模型限制, 由于在数据集中, 女性比专业人员(教师)的出现频率要更高, 所以导致翻译具有来自训练数据的偏见 bias
- Possible fix: 添加更多 profesore 的训练样本

- Error: "100,000 acres."
- Reason: 模型限制, 常识错误, hectáreas 表示公顷, acres 表示英亩 (acre的复数)。模型并未理解两个单位制之间的转换关系, 由于 acres 在训练集中的出现频率更高, 直接采用 acres 并且使用 hectáreas 附近的数字直接修饰 acres
- Possible fix: 添加关于 hectáreas 的训练数据

? Question 2.b

现在是时候探索您所训练的模型的输出了! 问题 1-i 中生成的模型的测试集翻译应该位于output /test_output.txt中。请找出你的模型产生的两个错误示例。你发现的两个例子应该是不同的错误类型, 并且与前一个问题中提供的例子不同。对于每个例子, 你应该:

- 写下西班牙语原文句子。源语句在 en_es_data/test.es 中
- 写下参考译文, 参考译文在en_es_data/test.en中
- 写下NMT模型的英文翻译, 模型翻译的句子位于output /test_output .txt中
- 识别NMT翻译中的错误
- 提供模型可能出错的原因(由于特定的语言构造或特定的模型限制)
- 描述一种可能的方法, 我们可以改变NMT系统, 以修复观察到的错误

Answer 2.b

- Source Sentence: El 5 de noviembre de 1990
- Reference Translation: On November 5th, 1990
- NMT Translation: On **five** of November 1990
- Error: five
- Reason: 模型限制, 模型没有数据集中充分学习到日期格式的转换
- Possible Fix: 增加更多关于西班牙语与英语之间的日期格式转换的数据样本

- Source Sentence: Y mis amigos hondureos me pidieron que dijera: "Gracias TED".
- Reference Translation: And my friends from Honduras asked me to say thank you, TED.
- NMT Translation: My friends were asked to say, "Thank you."
- Error: 说话的对象错误, 说话的人是我而不是我的朋友
- Reason: 句法结构有误并且有缺译现象
- Possible Fix: 尝试为模型的添加更有效的对齐方式, 如优化注意力模型

Question 2.c

BLEU评分是NMT系统中最常用的自动评价指标。它通常在整个测试集中计算, 但这里我们将考虑为单个示例定义的BLEU。假设我们有一个源句 s , 一组 k 个参考译文 r_1, \dots, r_k 和一个候选翻译 c 。为了计算 c 的BLEU分数, 我们首先为 c 计算修改后的 n-gram 精度 p_n , 对于 $n = 1, 2, 3, 4$:

$$p_n = \frac{\sum_{n\text{-gram} \in c} \min \left(\max_{i=1, \dots, k} \text{Count}_{r_i}(n\text{-gram}), \text{Count}_c(n\text{-gram}) \right)}{\sum_{n\text{-gram} \in c} \text{Count}_c(n\text{-gram})}$$

这里, 对于出现在候选翻译 c 中的每个 n-gram, 我们计算它在任何一个参考译文中出现的最大次数, 并以它出现在 c 中的次数为上限 (这是分子), 再除以 c 的 n-gram (分母)

接下来, 我们计算简洁代价 brevity penalty BP。令 c 作为 c 的长度, 让 r^* 作为最接近 c 的参考翻译的长度(在两个相等接近的参考翻译长度的情况下, 选择较短的参考翻译的长度作为 r^*)

$$BP = \begin{cases} 1 & \text{if } c \geq r^* \\ \exp \left(1 - \frac{r^*}{c} \right) & \text{otherwise} \end{cases}$$

最后，候选翻译 c 关于 r_1, \dots, r_k 的BLEU分数为：

$$BLEU = BP \times \exp\left(\sum_{n=1}^4 \lambda_n \log p_n\right)$$

其中， $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ 是总和为1的权重

Question 2.c.i

请考虑这个例子：

Source Sentence s : **el amor todo lo puede**

Reference Translation r_1 : *love can always find a way*

Reference Translation r_2 : *love makes anything possible*

NMT Translation c_1 : *the love can always do*

NMT Translation c_2 : *love can make anything possible*

分别计算 c_1, c_2 的BLEU分数。令 $\lambda_i = 0.5$ for $i \in \{1, 2\}$, $\lambda_i = 0$ for $i \in \{3, 4\}$ 。当计算BLEU分数时，显示你的计算过程(展示 p_1, p_2, c, r^*, BP 的计算值)。

根据BLEU评分，这两种NMT翻译中哪一种被认为是更好的翻译？你同意这是更好的翻译吗？

Answer 2.c.i

c_1

$$\begin{aligned} p_1 &= \frac{0+1+1+1+0}{5} = 0.6 \\ p_2 &= \frac{0+1+1+0}{4} = 0.5 \\ c &= 5 \\ r^* &= 4 \\ BP &= 1 \\ BLEU_{c_1} &= 1 * \exp(0.5 * \log(0.6) + 0.5 * \log(0.5)) = 0.5477 \end{aligned}$$

c_2

$$\begin{aligned} p_1 &= \frac{1+1+0+1+1}{5} = 0.8 \\ p_2 &= \frac{1+0+0+1}{4} = 0.5 \\ c &= 5 \\ r^* &= 4 \\ BP &= 1 \\ BLEU_{c_1} &= 1 * \exp(0.5 * \log(0.8) + 0.5 * \log(0.5)) = 0.632 \end{aligned}$$

根据 BLEU 分数， c_2 是得分更高的翻译，但我认为 c_1 的翻译更加好

Question 2.c.ii

我们的硬盘坏了，我们失去了参考翻译 r_2 。请重新计算 c_1 和 c_2 的BLEU分数，这次只针对 r_1 。两个NMT分一中，哪一个现在获得了更高的BLEU分数？你同意这是更好的翻译吗？

Answer 2.c.ii

c_1

$$\begin{aligned} p_1 &= \frac{0+1+1+1+0}{5} = 0.6 \\ p_2 &= \frac{0+1+1+0}{4} = 0.5 \\ c &= 5 \\ r^* &= 6 \\ BP &= \exp(1 - \frac{6}{5}) = 0.8187 \\ BLEU_{c_1} &= 0.8187 * \exp(0.5 * \log(0.6) + 0.5 * \log(0.5)) = 0.4484 \end{aligned}$$

c_2

$$\begin{aligned} p_1 &= \frac{1+1+0+0+0}{5} = 0.4 \\ p_2 &= \frac{1+0+0+0}{4} = 0.25 \\ c &= 5 \\ r^* &= 6 \\ BP &= \exp(1 - \frac{6}{5}) = 0.8187 \\ BLEU_{c_1} &= 0.8187 * \exp(0.5 * \log(0.4) + 0.5 * \log(0.25)) = 0.2589 \end{aligned}$$

根据 BLEU 分数， c_1 是得分更高的翻译，并且我认为这是对的

? Question 2.c.iii

由于数据可用性，NMT系统通常只根据一个参考翻译进行评估。请解释(用几句话)为什么这可能有问题？

Answer 2.c.iii

如果我们使用单一参考翻译，它增加了好翻译由于与单一参考翻译有较低的 n -gram overlap，而获得较差的BLEU分数的可能性。例如上例中，如果删去的参考翻译是 r_1 ，那么将使得 c_1 的BLEU分数变低。

如果我们增加更多的参考翻译，就会增加一个好翻译中 n -gram overlap 的几率，这样我们就有可能使好翻译获得相对较高的BLEU分数。

? Question 2.c.iv

列举了BLEU作为机器翻译的评价指标，相对于人工评价的两个优点和两个缺点。

Answer 2.c.iv

优点

- 自动评价，比人工评价更快，方便，快速
- BLEU的使用普及率较高，方便模型之间的效果对比

缺点

- 结果并不稳定，由于核心思想是 n -gram overlap，所以如果参考翻译不够丰富，会导致出现较好翻译获得较差BLEU分数的情况
- 不考虑语义与句法
- 不考虑词法，例如上例中的make和makes
- 未对同义词或相似表达进行优化

Reference

- [从SVD到PCA——奇妙的数学游戏](#)
- [alongstar518](#)
- [NLP 中评价文本输出都有哪些方法？为什么要小心使用 BLEU？](#)

评论