如何用深度学习做自然语言处理？这里有份最佳实践清单

2017-07-26 机器之心

选自ruder.io

机器之心编译

参与：机器之心编辑部

> *对于如何使用深度学习进行自然语言处理，本文作者 Sebastian Ruder 给出了一份详细的最佳实践清单，不仅包括与大多数 NLP 任务相关的最佳实践，还有最常见任务的最佳实践，尤其是分类、序列标注、自然语言生成和神经机器翻译。作者对最佳实践的选择很严格，只有被证明在至少两个独立的群体中有益的实践才会入选，并且每个最佳实践作者至少给出两个参引。作者承认这份清单并不全面，比如其不熟悉的解析、信息提取等就没有涉及。机器之心对该文进行了编译，原文链接请见文末。*

简介

本文是一系列关于如何使用神经网络进行自然语言处理（NLP）的最佳实践汇集，将随着新观点的出现定期进行更新，从而不断提升我们对用于 NLP 的深度学习的理解。

NLP 社区中有这样一句说法：带有注意力的 LSTM 能在所有任务上实现当前最佳的表现。尽管在过去的两年这确实是真的，NLP 社区却在慢慢偏离带有注意力的 LSTM，而去发现更有趣的模型。

但是，NLP 社区并非想再花费两年独立地（重新）发现下一个带有注意力的 LSTM。我们不打算重新发明已经奏效的技巧或方法。尽管现存的深度学习库已经从整体上编码了神经网络的最佳实践，比如初始化方案，但是很多其他的细节，尤其是特定任务或特定领域还有待从业者解决。

本文并不打算盘点当前最佳，而是收集与大量任务相关的最佳实践。换言之，本文并不描述某个特定架构，而是旨在收集那些构建成功框架的特征。其中的很多特征对于推动当前最佳是最有用的，因此我希望对于它们的更广泛了解将会带来更强的评估、更有意义的基线对比，以及更多灵感，帮助我们觉察那些可能奏效的方法。

本文假设你对神经网络应用于 NLP 的情况已经很熟悉（如果不熟悉，我建议你看一下 Yoav Goldberg 写的

A Primer on Neural Network Modelsfor Natural Language Processing：https://www.jair.org/media/4992/live-4992-9623-jair.pdf），并大体上对 NLP 或某个特定任务感兴趣。本文的主要目标是使你快速掌握相关的最佳实践，从而尽快做出有意义的贡献。我首先会对与绝大多数任务相关的最佳实践做一个概述，接着略述与最常见的任务相关的最佳实践，尤其是分类、序列标注、自然语言生成和神经机器翻译。

免责声明：把某些东西定义为最佳实践极其困难：最佳的标准是什么？如果有更好的实践出现呢？本文基于我的个人理解和经验（肯定不全面）。接下来，我将只讨论被证明在至少两个独立的群体中有益的实践。对于每个最佳实践我将给出至少两个参引。

最佳实践

词嵌入

在最近的 NLP 发展中，词嵌入无疑是最广为人知的最佳实践，这是因为预训练嵌入的使用对我们十分有帮助 (Kim, 2014) [12]。词嵌入的最佳维度绝大多数是依赖任务的：一个更小的维度更多在句法任务上工作更好，比如命名实体识别（named entity recognition）(Melamud et al., 2016) [44]，或者词性标注（POS) (Plank et al., 2016) [32]，尽管一个更大的维度对于更多的语义任务来说更有用，比如情感分析 (Ruder et al., 2016) [45]。

深度

虽然短时间内我们还无法达到计算机视觉的深度，但是 NLP 中的神经网络已经发展地更深了。现在最佳的方法通常使用 deep Bi-LSTM，它通常包含 3-4 层，比如词性标注 (Plank et al., 2016) 和语义角色标注 (He et al., 2017) [33]。一些任务的模型甚至更深。谷歌的 NMT 模型有 8 个编码器和 8 个解码器层，(Wu et al., 2016) [20]。然而，大多数情况下，模型超过 2 层所带来的性能提升是最小的 (Reimers & Gurevych, 2017) [46]。

这些观察适用于绝大多数序列标注和结构化预测问题。对于分类，深或者非常深的模型只在字符级的输入中表现良好，并且浅层的字词级模型依然是当前最佳 (Zhang et al., 2015; Conneau et al., 2016; Le et al., 2017) [28, 29, 30]。

层连接

训练深度神经网络时，可以使用一些核心技巧避免梯度消失问题。不同的层和连接因此被提出来了，这里我们将讨论 3 点：i) Highway 层，ii) 残差连接（residual connection），iii) 密集型残差连接。

Highway 层：它受到 LSTM 的门控机制所启发 (Srivastava et al., 2015) [1]。首先让我们假设一个单层的 MLP，它将一个非线性 g 的仿射变换应用到其输入 x：

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + b)$$

Highway 层接着计算以下函数：

$$\mathbf{h} = \mathbf{t} \odot g(\mathbf{W}_H\mathbf{x} + \mathbf{b}_H) + (1 - \mathbf{t}) \odot \mathbf{x}$$

其中 t=σ(WTx+bT) 被称作变换门（transform gate），(1–t) 被称作进位门（carry gate）。我们可以看到，Highway 层和 LSTM 门很相似，因为它们自适应地把输入的一些维度直接传递到输出。

Highway 层主要用于语言建模，并取得了当前最佳的结果 (Kim et al., 2016; Jozefowicz et al., 2016; Zilly et al., 2017) [2, 3, 4]，但它同时也用于其他任务，如语音识别 (Zhang et al., 2016) [5]。想了解更多相关信息和代码，可查看 Sristava 的主页（http://people.idsia.ch/~rupesh/very_deep_learning/）。

残差连接：残差连接（He et al., 2016）[6] 的首次提出是应用于计算机视觉，也是计算机视觉在 ImageNet 2016 夺冠的最大助力。残差连接甚至比 Highway 层更直接。我们使用代表当前层的指数 L 来增加之前的层输出 h。然后，残差连接学习以下函数：

$$\mathbf{h}^l = g(\mathbf{W}\mathbf{x}^l + \mathbf{b}) + \mathbf{x}^{l-1}$$

仅通过一个快捷连接，残差连接即可把之前层的输入添加到当前层。这一简单的更改缓解了梯度消失问题，因为

层级不能变得更好，模型可以默认使用恒等函数（identity function）。

密集型残差连接：密集型残差连接 (Huang et al., 2017) [7] (CVPR 2017 最佳论文奖）从每一个层向所有随后的层添加连接，而不是从每一个层向下一个层添加层：

$$\mathbf{h}^l = g(\mathbf{W}\mathbf{x}^l + \mathbf{b}) + \sum_{j=0}^{l} \mathbf{x}^j$$

密集型残差连接已成功应用于计算机视觉，也被证明在神经机器翻译方面的表现持续优于残差连接 (Britz et al., 2017) [27]。

Dropout

尽管在计算机视觉领域的多数应用中，批归一化已使其他正则化器变得过时，但是 dropout (Srivasta et al., 2014) [8] 依然是应用于 NLP 深度神经网络中的正则化器。0.5 的 dropout 率表明其在绝大多数场景中依然高效 (Kim, 2014)。近年来，dropout 的变体比如适应性 dropout（Ba & Frey, 2013） [9] 和进化 dropout (Li et al., 2016) [10] 已被提出，但没有一个在 NLP 社区中获得广泛应用。造成这一问题的主要原因是它无法用于循环连接，因为聚集 dropout masks 会将嵌入清零。

循环 dropout：循环 dropout （Gal & Ghahramani, 2016） [11] 通过在层 ll 的时间步中应用相同的 dropout masks 来解决这一问题。这避免了放大序列中的 dropout 噪音，并为序列模型带来了有效的正则化。循环 dropout 已在语义角色标注 (He et al., 2017) 和语言建模 (Melis et al., 2017) [34] 中取得了当前最佳的结果。

多任务学习

如果有额外的数据，多任务学习（MTL）通常可用于在目标任务中提升性能。

辅助目标（auxiliary objective）：我们通常能找到对我们所关心的任务有用的辅助目标 (Ruder, 2017) [13]。当我们已经预测了周围词以预训练词嵌入 (Mikolov et al., 2013) 时，我们还可以在训练中将其作为辅助目标 (Rei, 2017) [35]。我们也经常在序列到序列模型中使用相似的目标（Ramachandran et al., 2016）[36]。

特定任务层：尽管把 MTL 用于 NLP 的标准方法是硬参数共享，但允许模型学习特定任务层很有意义。这可通过把一项任务的输出层放置在较低级别来完成 (Søgaard & Goldberg, 2016) [47]。另一方法是诱导私有和共享的子空间 (Liu et al., 2017; Ruder et al., 2017) [48, 49]。

注意力机制

注意力机制是在序列到序列模型中用于注意编码器状态的最常用方法，它同时还可用于回顾序列模型的过去状态。使用注意力机制，系统能基于隐藏状态 s_1，...，s_m 而获得环境向量（context vector）c_i，这些环境向量可以和当前的隐藏状态 h_i 一起实现预测。环境向量 c_i 可以由前面状态的加权平均数得出，其中状态所加的权就是注意力权重 a_i：

$$\mathbf{c}_i = \sum_j a_{ij}\mathbf{s}_j$$
$$\mathbf{a}_i = \mathrm{softmax}(f_{att}(\mathbf{h}_i, \mathbf{s}_j))$$

注意力函数 f_att(h_i,s_j) 计算的是目前的隐藏状态 h_i 和前面的隐藏状态 s_j 之间的非归一化分配值。在下文中，我们将讨论四种注意力变体：加性注意力（additive attention）、乘法（点积）注意力（multiplicative attention）、自注意力（self-attention）和关键值注意力（key-value attention）。

加性注意力是最经典的注意力机制 (Bahdanau et al., 2015) [15]，它使用了有一个隐藏层的前馈网络来计算注意力的分配：

$$f_{att}(\mathbf{h}_i, \mathbf{s}_j) = \mathbf{v}_a{}^\top \tanh(\mathbf{W}_a[\mathbf{h}_i; \mathbf{s}_j])$$

其中 v_a 和 W_a 是所学到的注意力参数，[* ; *] 代表了级联。类似地，我们同样能使用矩阵 W_1 和 W_2 分别为 h_i 和 s_j 学习单独的转换，这一过程可以表示为：

$$f_{att}(\mathbf{h}_i, \mathbf{s}_j) = \mathbf{v}_a{}^\top \tanh(\mathbf{W}_1\mathbf{h}_i + \mathbf{W}_2\mathbf{s}_j)$$

乘法注意力（Multiplicative attention）(Luong et al., 2015) [16] 通过计算以下函数而简化了注意力操作：

$$f_{att}(h_i, s_j) = h_i^\top \mathbf{W}_a s_j$$

加性注意力和乘法注意力在复杂度上是相似的，但是乘法注意力在实践中往往要更快速、具有更高效的存储，因为它可以使用矩阵操作更高效地实现。两个变体在低维度 d_h 解码器状态中性能相似，但加性注意力机制在更高的维度上性能更优。缓解这一现象的方法是将 f_att(h_i,s_j) 缩放到 d_h^(-1/2) 倍 (Vaswani et al., 2017) [17]。

注意力机制不仅能用来处理编码器或前面的隐藏层，它同样还能用来获得其他特征的分布，例如阅读理解任务中作为文本的词嵌入 (Kadlec et al., 2017) [37]。然而，注意力机制并不直接适用于分类任务，因为这些任务并不需要情感分析（sentiment analysis）等额外的信息。在这些模型中，通常我们使用 LSTM 的最终隐藏状态或像最大池化和平均池化那样的聚合函数来表征句子。

自注意力机制（Self-attention）通常也不会使用其他额外的信息，但是它能使用自注意力关注本身进而从句子中抽取相关信息 (Lin et al., 2017) [18]。自注意力又称作内部注意力，它在很多任务上都有十分出色的表现，比如阅读理解 (Cheng et al., 2016) [38]、文本继承 (textual entailment/Parikh et al., 2016) [39]、自动文本摘要 (Paulus et al., 2017) [40]。

我们能计算每个隐藏状态 h_i 的非归一化分配值从而简化加性注意力：

$$f_{att}(\mathbf{h}_i) = \mathbf{v}_a{}^\top \tanh(\mathbf{W}_a \mathbf{h}_i)$$

在矩阵形式中，对于隐藏状态 H=h_1,...,h_n，我们能通过以下形式计算注意力向量 a 和最后的句子表征 c：

$$\mathbf{a} = \text{softmax}(\mathbf{v}_a \tanh(\mathbf{W}_a \mathbf{H}^\top))$$
$$\mathbf{c} = \mathbf{H}\mathbf{a}^\top$$

我们不仅可以抽取一个向量，同时还能通过将 v_a 替代为 V_a 矩阵而执行一些其他注意力特征，这可以令我们抽取注意力矩阵 A：

$$\mathbf{A} = \text{softmax}(\mathbf{V}_a \tanh(\mathbf{W}_a \mathbf{H}^\top))$$
$$\mathbf{C} = \mathbf{A}\mathbf{H}$$

在实践中，我们可以执行以下的正交约束而惩罚计算冗余，并以 Frobenius 范数平方的形式鼓励注意力向量的多样性：

$$\Omega = \|(\mathbf{A}\mathbf{A}^\top - \mathbf{I}\|_F^2$$

Vaswani et al. (2017) 同样使用了类似的多头注意力（multi-head attention）。

最后，关键值注意力 (Daniluk et al., 2017) [19] 是最近出现的注意力变体机制，它将形式和函数分开，从而为注意力计算保持分离的向量。它同样在多种文本建模任务 (Liu & Lapata, 2017) [41] 中发挥了很大的作用。具体来说，关键值注意力将每一个隐藏向量 h_i 分离为一个键值 k_i 和一个向量 v_i：[k_i;v_i]=h_i。键值使用加性注意力来计算注意力分布 a_i：

$$\mathbf{a}_i = \text{softmax}(\mathbf{v}_a{}^\top \tanh(\mathbf{W}_1[\mathbf{k}_{i-L}; \ldots; \mathbf{k}_{i-1}] + (\mathbf{W}_2 \mathbf{k}_i)\mathbf{1}^\top))$$

其中 L 为注意力窗体的长度，l 为所有单元为 1 的向量。然后使用注意力分布值可以求得环境表征 c_i：

$$\mathbf{c}_i = [\mathbf{v}_{i-L}; \ldots; \mathbf{v}_{i-1}]\mathbf{a}^\top$$

其中环境向量 c_i 将联合现阶段的状态值 v_i 进行预测。

最优化

最优化算法和方案通常是模型的一部分，并且常常被视为黑箱操作。有时算法轻微的变化，如在 Adam 算法中减少超参数β2 的值 (Dozat & Manning, 2017) [50] 将会造成优化行为的巨大改变。

Adam 方法 (Kingma & Ba, 2015) [21] 是使用最广泛、最常见的优化算法，它通常也作为 NLP 研究员的优化器。Adam 方法要明显地比 vanilla 随机梯度下降更优秀，并且其收敛速度也十分迅速。但近来有研究表明通过精调并带动量的梯度下降方法要比 Adam 方法更优秀 (Zhang et al., 2017) [42]。

从优化方案来说，因为 Adam 方法会适应性地为每一个参数调整学习速率 (Ruder, 2016) [22]，所以我们可以使用 Adam 方法精确地执行 SGD 风格的退火处理。特别是我们可以通过重启（restart）执行学习速率退火处理：即设定一个学习速率并训练模型，直到模型收敛。然后，我们可以平分学习速率，并通过加载前面最好的模型而重启优化过程。在 Adam 中，这会令优化器忘记预训练参数的学习速率，并且重新开始。Denkowski & Neubig (2017) [23] 表示带有两个重启和学习速率退火处理的 Adam 算法要比带有退火处理的 SGD 算法更加优秀。

集成方法

通过平均多个模型的预测将多个模型组合为一个集成模型被证明是提高模型性能的有效策略。尽管在测试时使用集成做预测十分昂贵，最近提取方面的一些进展允许我们把昂贵的集成压缩成更小的模型 (Hinton et al., 2015; Kuncoro et al., 2016; Kim & Rush, 2016) [24, 25, 26]。

如果评估模型的多样性增加 (Denkowski & Neubig, 2017)，集成是确保结果可靠的重要方式。尽管集成一个模型的不同检查点被证明很有效 (Jean et al., 2015; Sennrich et al., 2016) [51, 52]，但这种方法牺牲了模型的多样性。周期学习率有助于缓解这一影响 (Huang et al., 2017) [53]。但是，如果资源可用，我们更喜欢集成多个独立训练的模型以最大化模型多样性。

超参数优化

我们可以简单地调整模型超参数从而在基线上获得显著提升，而不仅仅只是使用预定义或现有的超参数来训练模型。最近 Bayesian Optimization 的新进展可以用于在神经网络黑箱训练中优化超参数 (Snoek et al., 2012) [56]，这种方法要比广泛使用的网格搜索高效地多。LSTM 的自动超参数调整已经在语言建模产生了最佳的性能，远远胜过其他更复杂的模型 (Melis et al., 2017)。

LSTM 技巧

学习初始状态：我们通常初始化 LSTM 状态为零向量。但我们可以将初始状态看作参数进行优化，而不是人为地调整来提升性能。这一方法十分受 Hinton 的推荐。关于这一技巧的 TensorFlow 实现，详见：https://r2rt.com/non-zero-initial-states-for-recurrent-neural-networks.html

尝试输入和输出嵌入：适合于输入和输出嵌入在 LSTM 模型中占了绝大多数参数数量的情况。如果 LSTM 在语言建模中预测词汇，输入和输出参数可以共享 (Inan et al., 2016; Press & Wolf, 2017) [54, 55]。这一技巧在不允许学习大规模参数的小数据集中十分有用。

梯度范数截断（Gradient norm clipping）：降低梯度消失风险的一个方法是截断其最大值 (Mikolov, 2012) [57]。但是这并没有持续提升性能（Reimers & Gurevych, 2017）。与其独立地截断每个梯度，截断梯度的全局范数 (Pascanu et al., 2013) 反而会带来更加显著的提升（这里有一个 Tensorflow 实现：https://stackoverflow.com/questions/36498127/how-to-effectively-apply-gradient-clipping-in-tensor-flow）。

下投影（Down-projection）：为了进一步减少输出参数的数量，LSTM 的隐态可以被投影到更小的尺寸。这对带有大量输出的任务尤其有用，比如语言建模（Melis et al., 2017）。

特定任务的最佳实践

下面，我们要介绍特定任务的最佳实践。大部分模型在执行特定类型的单项任务时表现很好，部分模型可以应用于其他任务，不过在应用之前需要验证其性能。我们还将讨论以下任务：分类、序列标注、自然语言生成（NLG）和自然语言生成的特殊案例神经机器翻译。

分类

由于卷积操作更加高效，近期 CNN 应用范围扩大，成为处理 NLP 中分类任务的通用方法。下面的最佳实践和 CNN 相关，可选择多个最优超参数（optimal hyperparameter）。

- CNN 过滤器：使过滤器大小接近最优过滤器大小，如 (3,4,5) 性能最佳（Kim, 2014; Kim et al., 2016）。特征映射的最佳数量范围是 50~600（Zhang & Wallace, 2015）[59]。
- 聚合函数（Aggregation function）：1- 最大池化优于平均池化和 k- 最大池化（Zhang & Wallace, 2015）。

序列标注

序列标注在 NLP 中非常普遍。现有的很多最佳实践都是模型架构的一个环节，下列指南主要讨论模型输出和预测阶段。

标注策略（Tagging scheme）： 对于将标签分配到文本分隔的任务，不同的标注策略均可采用。比如：BIO，分隔的第一个符号处标注 B-tag，其他符号处标注 I-tag，分隔外的符号标注 O-tag；IOB，和 BIO 相似，不同之处在于如果前面的符号属于相同的类别，但不属于同一个分隔，则使用 B-tag 来标注；IOBES，还需要在单个符号实体处标注 S-tag，每个分隔的最后一个符号处标注 E-tag。IOBES 和 BIO 的性能相似。

条件随机场输出层（CRF output layer）： 如果输出之间存在依赖，如在命名实体识别（named entity recognition）中，可以用线性链条件随机场（linear-chain conditional random field）代替最后的 softmax 层。经证实，这种方法对于要求对约束进行建模的任务有持续改善的效果（Huang et al., 2015; Max & Hovy, 2016; Lample et al., 2016）[60, 61, 62]。

约束解码（Constrained decoding）： 除了条件随机场输出层以外，还可用约束解码来排除错误排序，即不产生有效的 BIO 过渡（BIO transition）（He et al., 2017）。约束解码的优势在于可以执行随意约束（arbitrary constraint），如特定任务约束或句法约束。

自然语言生成

多数现有最佳实践可用于自然语言生成（NLG）。事实上，目前出现的很多技巧都植根于语言建模方面的进步，语言建模是最典型的 NLP 任务。

建模覆盖率（Modelling coverage）： 重复是 NLG 任务的一大难题，因为当前的模型无法很好地记忆已经产生的输出结果。在模型中直接设置建模覆盖率是解决该问题的好方法。如果提前知道哪些实体应该出现在输出结果中（就像菜谱中的调料），则需要使用一个检查表（checklist）（Kiddon et al., 2016）[63]。如果使用注意力机制，我们可以追踪覆盖率向量 c_i，它是过去的时间步上注意力分布 a_t 的总和（Tu et al., 2016; See et al., 2017）[64, 65]：

$$\mathbf{c}_i = \sum_{t=1}^{i-1} \mathbf{a}_t$$

该向量可以捕捉我们在源语言所有单词上使用的注意力。现在我们可以在覆盖率向量上设置加性注意力（additive attention），以鼓励模型不重复关注同样的单词：

$$f_{att}(\mathbf{h}_i, \mathbf{s}_j, \mathbf{c}_i) = \mathbf{v}_a{}^\top \tanh(\mathbf{W}_1\mathbf{h}_i + \mathbf{W}_2\mathbf{s}_j + \mathbf{W}_3\mathbf{c}_i)$$

此外，我们可以添加辅助损失（auxiliary loss），该损失可以捕捉我们想关注的特定任务的注意力行为：我们希望神经机器翻译可以做到一对一对齐（one-to-one alignment）；如果最后的覆盖率向量多于或少于每一个指数上的覆盖率向量，那么模型将被罚分（Tu et al., 2016）。总之，如果模型重复处理同样的位置，我们就会惩罚该模型（See et al., 2017）。

神经机器翻译

- 虽然神经机器翻译只是 NLG 的一个分支，但 NMT 获得了大量关注，有许多方法专门为该任务开发。相似地，许多最佳实践或超参数选择只能应用到 NMT 领域。
- 嵌入维度（Embedding dimensionality）：2048 维嵌入的性能最佳，但很少达到该效果。128 维嵌入的性能却出乎意料地好，收敛速度几乎达到之前的 2 倍（Britz et al., 2017）。
- 编码器和解码器深度：编码器的深度无需超过 2–4 层。深层模型性能优于浅层模型，但多于 4 层对解码器来说没有必要（Britz et al., 2017）。
- 方向性（Directionality）：双向编码器性能稍好于单向编码器。Sutskever et al.（2014）[_67_] 提出颠倒源语言的顺序，以减少长期依赖的数量。使用单向编码器颠倒源语言顺序优于未颠倒语序（Britz et al., 2017）。
- 束搜索策略（Beam search strategy）：大小 10、长度归一化罚项为 1.0 的中型束（Wu et al., 2016）性能最佳（Britz et al., 2017）。
- 子词翻译（Sub-word translation）：Senrich et al. (2016) [66] 提出根据字节对编码（byte-pair encoding / BPE）将单词分隔成子词（sub-word）。BPE 迭代合并出现频率高的符号对（symbol pair），最后将出现频率高的 n 元合并成一个单独的符号，进而有效去除非词表词（out-of-vocabulary-word）。该技术最初用来处理罕见单词，但是子词单元的模型性能全面超过全词系统，32000 个子词单元是最高效的单词数

量（Denkowski & Neubig, 2017）。

结语

我确定这份清单上一定有遗漏的最佳实践。相似地，也有很多我不熟悉的任务，如解析、信息提取，我没办法做出推荐。我希望本文对开始学习新的 NLP 任务有所帮助。即使你对这里列出的大部分内容都很熟悉，我也希望你能够学到一些新的东西或者重新掌握有用的技巧。SYNCED

参考：

因参考文献过多，超出字数限制，请点击以下原文查看。

原文链接：http://ruder.io/deep-learning-nlp-best-practices/index.html#introduction