本科毕业论文外文翻译

文献原文:

Kim Y, Jernite Y, Sontag D, et al. Character-aware neural language models[J]. arXiv preprint arXiv:1508.06615, 2015.

以字为单位的自然语言模型

摘要

我们描述了一个只依赖以字为单位输入的自然语言模型。我们的预测依然基于以词为单位。我们的模型对于字使用了卷积神经网络 (CNN) 和高速网络，模型的输出将作为输入，输入到由长短期记忆人工神经网络神经元（LSTM) 组成的循环神经网络 (RNN)。对于 Penn Treebank 英文数据集，我们的模型比现存的主流模型的参数少 60%，却仍然能取得不相上下多结果。对于内含丰富词法的语言（比如阿拉伯语、捷克语、法语、德语西班牙语、俄语），我们的模型的在需要很少的参数的情况下，表现超过 LSTM 的基准线。我们的结果表明，对于很多语言来说，以字为单位的输入，已经足够对语言建模。进一步分析由我们模型的产生的词的表示表明，我们的模型能够基于字编码并且包含语义信息。

1 介绍

语言建模是人工智能和自然语言处理最基础的任务。一个语言模型是对于一系列词的概率分布和转换方法。其中，转换方法通常包括通过计数和子序列拟合对马尔科夫链第 n 次序的假设和估测 n-gram 的概率。基于计数的模型比较容易训练，但是对于稀有词，可能由于数据稀疏而不能很好的估计。

神经语言模型为了解决 n-gram 数据稀疏问题通过参数把词向量化（词嵌入），并把向量词作为神经网络的输入。向量化所使用的参数在训练中学习到。通过神经语言模型获得的词嵌入具有语义上比较接近的词在向量空间比较接近这一性质。

虽然神经语言模型的表现优于计数语言模型，但是该模型不能表示语素信息。比如，该模型不能发现相同前缀的词（例如 eventfull,eventfully,uneventful,uneventfully）是结构上有关联的，他们应该在向量空间上结构相关。因此，稀有词的嵌入可能被错误的

估计。这个问题对于词法丰富的语言来说，问题尤其严重
在这篇文章中，我们提出了一个语言模型，这个模型通过以字为单位的卷积神经网络来获取子词信息。模型的输出将作为 RNN 语言模型的输入。不同于之前使用词素来获取子词信息的模型，我们的模型不需要对词素标记。同样，不同于近期提出的结合词嵌入和字节的模型，我们的模型在输入层不使用词嵌入。因为对于神经语言模型，大多数的参数都是用于生产词嵌入，我们的模型所需的参数相比于之前的模型显著减少。特别适用于对于模型大小有限制的平台。
作为总结，我们的贡献主要有下面几点：

- 对于英语，在 Penn Treebank 英文数据集上，我们的模型比现存的主流模型的参数少 60%，却仍然能取得不相上下多结果
- 对于内含丰富词法的语言（比如阿拉伯语、捷克语、法语、德语、西班牙语、俄语），我们的模型的在需要很少的参数的情况下，表现超过 LSTM 的基准线

2 模型

Figure 1 直观的显示了我们模型的架构。经典的神经语言模型采用词嵌入作为输入，我们的模型采用单层字节 max-over-time 池化卷积神经网络的输出作为输入
作为记号，我们用粗体小写字母标记向量，粗体大写字母标记矩阵，斜体小写字母标记变量，花体大写字母标记集合。为了标记方便，我们假设字和词都已经转换成了索引。

2.0.1 递归神经网络

递归神经网络（RNN）特别适合对于时序模型建模。在每一个时步 t，根据输入向量 $\mathbf{x_i} \in \mathbb{R}^\mathbf{n}$、隐式状态 $\mathbf{h_i}$，应用下面的公式产生下一层隐式状态:

$$\mathbf{h_i} = \mathbf{f}((\mathbf{Wx})_\mathbf{i} + \mathbf{Uh_{t-1}} + \mathbf{b}) \tag{1}$$

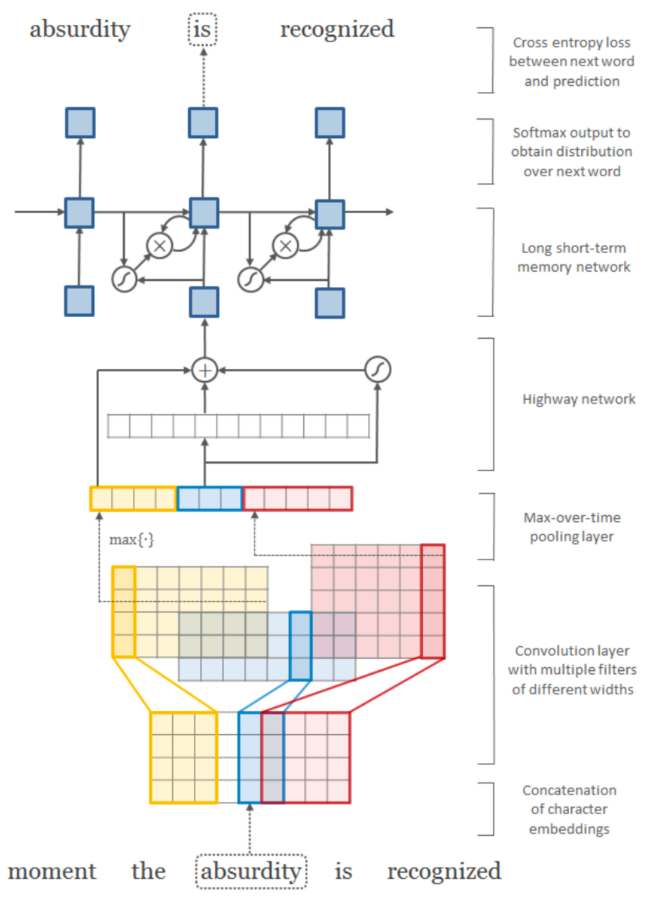
absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized

图 1　模型架构

在这里 $\mathbf{W} \in \mathbb{R}^{m\times n}, \mathbf{U} \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}_m$ 都是仿射变换的参数，f 是非线性对每个元素的函数。理论上，RNN 中的隐式状态 $\mathbf{h_t}$ 包含了 t 个时步的历史信息。然而在实际中，由于权重指数级爆炸或消失的问题，基本的 RNN 难以捕捉长期时间关联。

长短期记忆（LSTM）通过将 RNN 在每一时步增加一个记忆单元向量 $\mathbf{c_i} \in \mathbb{R}_n$ 来解决这一问题。具体来讲，LSTM 的每一步利用输入 $\mathbf{x_i}, \mathbf{h_{t-1}}, \mathbf{c_{t-1}}$ 根据下面的公式，计算出 $\mathbf{h_t}, \mathbf{c_t}$:

$$\mathbf{i_t} = \sigma(\mathbf{W^i x_i} + \mathbf{U^i h_{t-1}} + \mathbf{b^i}) \tag{2}$$

$$\mathbf{f_t} = \sigma(\mathbf{W^f x_t} + \mathbf{U^f h_{t-1}} + \mathbf{b^f}) \tag{3}$$

$$\mathbf{o_i} = \sigma(\mathbf{W^o x_t} + \mathbf{U^o h_{t-1}} + \mathbf{b_o}) \tag{4}$$

$$\mathbf{g_t} = \mathbf{tanh}(\mathbf{W^g x_t} + \mathbf{U^g h_{t-1}} + \mathbf{b_g}) \tag{5}$$

$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{g_t} \tag{6}$$

$$\mathbf{h_t} = \mathbf{o_t} \odot \mathbf{tanh}(\mathbf{c_t}) \tag{7}$$

在这里 $\sigma(\cdot)$ 和 $tanh(\cdot)$ 是对每个元素的 sigmoid 函数和双曲正切函数,$\odot$ 是元素间相乘运算符,$\mathbf{i_t}, \mathbf{f_t}, \mathbf{o_t}$ 分别是输入门、遗忘门、输出门。当 $t = 1$ 时,$\mathbf{h_0}$ 和 $\mathbf{c_0}$ 被初始化为零向量。LSTM 需要的参数是 $\mathbf{W_j}, \mathbf{U_j}, \mathbf{b_j}, \mathbf{j} \in \{\mathbf{i}, \mathbf{f}, \mathbf{o}, \mathbf{g}\}$

LSTM 中的记忆单元是随时间变化的,从而缓解了权重指数消失问题。虽然权重指数爆炸仍然是一个问题,但是在实际训练过程中,一些简单的优化策略(例如变化率截断)效果很好。LSTM 在包括语言建模在内的很多任务中显示出比基本 RNN 更好的结果。通过使 $\mathbf{h_t}$ 作为 t 时的输入,可以很容易的将 RNN / LSTM 扩展到多层。实际上,在很多任务中,使用多层网络是获得有竞争力结果的关键选择

2.1 Recurrent Neural Network Language Model

记 $\mathscr{V}$ 为固定的词汇集大小。给定历史序列 $w_{1:t} = [w_1, ..., w_t]$, 一个语言模型确定了 w_{t+1} 的概率分布。一个循环神经网络语言模型(RNN-LM) 通过对隐式层应用仿射变换然后再应用 softmax 函数来实现这一点:

$$Pr(w_{t+1} = j \mid w_{1:t}) = \frac{exp(\mathbf{h_t} \cdot \mathbf{p^j} + \mathbf{q^j})}{\sum_{j' \in \mathscr{V}} exp(\mathbf{h_t} \cdot \mathbf{p^{j'}} + \mathbf{q^{j'}})} \tag{8}$$

在这里 $\mathbf{p^j}$ 是 $\mathbf{P} \in \mathbb{R}^{\mathbf{m} \times |\mathscr{V}|}$ 的第 j 列,q^j 是偏差项。同样的,传统的 RNN 语言模型把词作为输入,如果 $w_t = k$,传统模型在时步 t 的输入是 $\mathbf{X} \in \mathbb{R}^{\mathbf{n} \times |\mathscr{V}|}$ 的第 k 列 x^k。我们的模型把词嵌入矩阵 $\mathbf{X}$ 替换成了下面描述的字为单位的卷积神经网络的输出。

我们记 $w_{1:T} = [w_1, ..., w_T]$ 为训练词库的词序列,训练包括最小化该序列的负 log 概率 (NLL), 这一过程通常有啊截断反向传播来实现

$$NLL = -\sum_{t=1}^{T} logPr(w_t \mid w_{1:t-1}) \tag{9}$$

2.1.1　字单位的卷积神经网络

在我们的模型中，t 时刻的输入是字单位的卷积神经网络的输出（CharCNN), 我们将在这里小节里面描述 CharCNN。CNN 在计算机视觉领域的应用已经取得了突出的效果，并且对于多种自然语言处理（NLP）任务也是十分有效的。针对 NLP 应用 CNN 有不同的架构，因为 NLP 需要时序卷积而不是空间卷积。
令 $\mathscr{C}$ 为字的集合，d 为字嵌入的维度，$\mathbf{Q} \in \mathbb{R}^{\mathbf{d}\times|\mathscr{C}|}$ 为字嵌入矩阵。假设词 $k \in \mathscr{V}$ 是由字序列 $[c_1, ..., c_l]$ 组成，l 是词 k 的长度。k 的字表示由矩阵 $\mathbf{C^k} \in \mathbb{R}^{\mathbf{d\times l}}$ 给出，该矩阵的第 j 列是字 c_j 的字嵌入
我们对 $\mathbf{C^k}$ 和宽度为 w 的过滤器 $\mathbf{H} \in \mathbb{R}^{\mathbf{d\times w}}$ 应用窄卷积。然后，我们加上偏差，再应用一个非线形函数来获得一个特征图谱 $\mathbf{f^k} \in \mathbb{R}^{\mathbf{l-w+1}}$。$\mathbf{f^k}$ 的第 i 项由下式得出：

$$\mathbf{f^k[i]} = \mathbf{tanh}(\langle \mathbf{C^k}[*, \mathbf{i : i + w - 1}], \mathbf{H}\rangle + \mathbf{b}) \tag{10}$$

在这里 $bfC^k[*, i : i+w-1]$ 是 $\mathbf{C^k}$ 的 i 到 $i+w-1$ 列，$\langle \mathbf{A}, \mathbf{B} = \mathbf{Tr}(\mathbf{AB^T})\rangle$ 是 Frobenius 内积。最后我们应用 max-over-time 作为过滤器 $\mathbf{H}$ 对应的特征

$$y^k = max_i \mathbf{f^k[i]} \tag{11}$$

这样做的目的是对于一个给定的过滤器，获取最重要的特征，也就是具有最高值的特征。一个过滤器本质上是选出一个字 n 元语法，n 的大小由过滤器宽度决定。
我们通过一个特征由一个过滤矩阵得出描述了这个过程。我们的 CharCNN 使用多个不同宽度的过滤器获得 k 的特征向量。所以如果我们有 h 个过滤器 $\mathbf{H_1}, ...\mathbf{H_h}$, 就有 k 的输入表示 $\mathbf{y^k} = [\mathbf{y^k_1}, ..., \mathbf{y^k_h}]$，对于很多 NLP 应用 h 通常在 $[100, 1000]$ 这个范围内选择。

2.2　高速网络

我们只需要在 RNN-LM 中在 t 时刻，用 $\mathbf{y^k}$ 替换 $\mathbf{x^k}$，正如我们稍后将要证明的，这样就已经能有很高效的表示了。我们也可以对 $\mathbf{y^k}$ 使用一个多层感受器（MLP), 来

建模过滤器选择的 n 元语法的交集，但是我们发现这样的表现不是很好。

我们通过将 $\mathbf{y^k}$ 运行在高速网络上进一步提高的我们的表现。高速网络是一层的 MLP 应用了仿射变换之后再施加一个非线形函数，从而得到一组新的特征。

$$\mathbf{z} = \mathbf{g}(\mathbf{W}\mathbf{y} + \mathbf{b}) \tag{12}$$

一层高速网络实现了做了下面的过程：

$$\mathbf{z} = \mathbf{t} \odot \mathbf{g}(\mathbf{W_H}\mathbf{y} + \mathbf{b_H}) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{y} \tag{13}$$

在这里 g 是一个非线性函数,$\mathbf{t} = \sigma(\mathbf{W_T}\mathbf{y} + \mathbf{b_T})$ 被称作转换门，$(\mathbf{1} - \mathbf{t})$ 被称作进位门。与 LSTM 网络中的记忆单元相似，高速网络会使一些维度的输入直接作为输出。在构建的时候，$\mathbf{y}$ 和 $\mathbf{z}$ 的维度必须相同，$\mathbf{W_T}$ 和 $\mathbf{W_T}$ 都是方阵。

3 实验设置

根据语言建模的标准，我们使用复杂度（PPL) 来衡量我们模型的表现。对于一个序列 $[w_1, ..., w_T]$，一个模型的复杂度由下式给出：

$$PPL = exp(\frac{NLL}{T}) \tag{14}$$

在这里 NLL 是对测试集来计算的。我们针对不同的语言和不同大小的词汇集进行了测试。

针对 Penn Treebank（PTB）数据集，我们进行了高阶参数查询、模型检查和分离学习。使用 0-20 进行标准学习、21-22 进行验证、23-24 进行测试。PTB 有一百万个标签，$|\mathscr{V}| = 10k$, 已经被广泛的用于语言建模。

在针对 PTB 数据集优化高阶参数之后，我们将模型应用于很多词法丰富的语言：捷克语、德语、法语、西班牙语、俄语、阿拉伯语。尽管未经处理的数据是公开可用的，我们从作者那里获取到了预处理过的数据，他们的 NLM 充当了我们模型的基准。我们在每个语言有一百万个标签的小数据集（DATA-S）和 $|\mathscr{V}| = 10k$ 远大于 PTB 的大数据集（DATA-L）上都进行了训练。

在这些数据集中，只有只出现一次的词汇会被 <unk> 替换，所以我们有效的使用了

数据集上包含的词汇集。

	DATA-S			DATA-L		
	$\|\mathcal{V}\|$	$\|\mathcal{C}\|$	T	$\|\mathcal{V}\|$	$\|\mathcal{C}\|$	T
English (EN)	10 k	51	1 m	60 k	197	20 m
Czech (CS)	46 k	101	1 m	206 k	195	17 m
German (DE)	37 k	74	1 m	339 k	260	51 m
Spanish (ES)	27 k	72	1 m	152 k	222	56 m
French (FR)	25 k	76	1 m	137 k	225	57 m
Russian (RU)	62 k	62	1 m	497 k	111	25 m
Arabic (AR)	86 k	132	4 m	–	–	–

图 2　词汇集统计数据

3.1　优化

我们使用截断反向传播来训练我们的模型。我们使用 stochastic gradient descent 反向传播 35 个时步。学习率初始设为 1.0，当 PPL 没有在每个 epoch 在验证集下降超过 1.0 时，学习率变为原来的一半。

在 DATA-S 上，我们令 batch 大小为 20，在 DATA-L 中我们使用 batch 大小为 100。变化率是每个 batch 的平均。我们对于非阿拉伯语训练 25 个 epoch，对于阿拉伯语训练 30 个 epoch，选出在验证集上表现最好的模型。模型随机初始化为服从 [-0.05,0.05] 上的均匀分布。

我们在 LSTM 从输入到隐式层和 softmax 到输出层使用概率为 0.5 的丢弃法进行正规化学习。我们进一步限制变化率的范数小于 5。如果 L_2 规范化的变化率超过 5，在更新之前，我们设置他为 5。变化率范数限制对于训练模型是很重要的。我们的选择很大程度上是受 Zaremba et al. 做的工作的启发。

最后，为了加速 DATA-L 上的学习，我们使用了 hierarchical softmax。Hierarchical softmax 是一个训练 $|\mathscr{V}|$ 大小不同的语言模型的常用策略。我们选择 $c = \lceil\sqrt{|\mathscr{V}|}\rceil$ 个簇，随机的把 $\mathscr{V}$ 分成大小相同的子集 $\mathscr{V}_1, ..., \mathscr{V}_c$，这样就有：

$$Pr(w_{t+1} = j \mid w_{1:t}) = \frac{exp(\mathbf{h_t} \cdot \mathbf{s^r} + \mathbf{t^r})}{\sum_{r'=1} cexp(\mathbf{h_t} \cdot \mathbf{s^{r'}} + \mathbf{t^{r'}})} \times \frac{exp(\mathbf{h_t} \cdot \mathbf{p_r^j} + \mathbf{q_r^j})}{\sum_{j' \in \mathscr{V}_r} exp(\mathbf{h_t} \cdot \mathbf{p_r^{j'}} + \mathbf{q_r^{j'}})} \qquad (15)$$

在这里 r 是簇的索引，有 $j \in \mathcal{V}_r$。上式的第一项是选择第 r 簇的概率，第二项是已知第 r 簇被选的情况下选中词 j 的概率。

4 实验结果

4.1 English Penn Treebank

		Small	Large
CNN	d	15	15
	w	$[1, 2, 3, 4, 5, 6]$	$[1, 2, 3, 4, 5, 6, 7]$
	h	$[25 \cdot w]$	$[\min\{200, 50 \cdot w\}]$
	f	tanh	tanh
Highway	l	1	2
	g	ReLU	ReLU
LSTM	l	2	2
	m	300	650

图 3 小模型和大模型的架构

我们为了评估性能和大小的取舍，对我们的模型训练了两个版本。小模型（LSTM-Char-Small）和大模型 (LSTM-Char-Large) 的架构在表 2 中进行了总结。
和其他基准一样，我们也用词嵌入训练了两个比较模型（LSTM-Word-Small,LSTM-Word-Large)。LSTM-Word-Small 使用了 200 个隐式单元。LSTM-Word-Large 使用了 650 个隐式单元。词嵌入的大小分别是 200 和 650. 这些大小的选择是为了保持和字单位模型相似的参数数目。
正如从表 3 中看到的，我们的模型在参数少 60% 的情况下，与现在领先的模型取得差不多的表现。我们的小模型明显优于其他大小相似的 NLM。

4.2 其他语言

我们模型在 English PTB 上的优异表现表明可以使用它来处理更大规模大数据。然而，英语从词法角度讲是相对简单的，所以我们下一部分的结果主要集中在有更丰富词法的语言上。

	PPL	Size
LSTM-Word-Small	97.6	5 m
LSTM-Char-Small	92.3	5 m
LSTM-Word-Large	85.4	20 m
LSTM-Char-Large	78.9	19 m
KN-5 (Mikolov et al. 2012)	141.2	2 m
RNN† (Mikolov et al. 2012)	124.7	6 m
RNN-LDA† (Mikolov et al. 2012)	113.7	7 m
genCNN† (Wang et al. 2015)	116.4	8 m
FOFE-FNNLM† (Zhang et al. 2015)	108.0	6 m
Deep RNN (Pascanu et al. 2013)	107.5	6 m
Sum-Prod Net† (Cheng et al. 2014)	100.0	5 m
LSTM-1† (Zaremba et al. 2014)	82.7	20 m
LSTM-2† (Zaremba et al. 2014)	78.4	52 m

图 4 小模型和大模型的架构

我们将我们的结果同 morphological logbilinear (MLBL) 模型相比较。MLBL 通过在输入和输出阶段的语素嵌入也考虑了子词信息。因为在同 MLBL 比较时，我们使用的 LSTM 比 MLBL 使用的 feed-forward 有众所周知的更优异的表现，我们又训练了一个 LSTM 版本的语素 NLM。这个版本中，传入 LSTM 的词表示是词的语素嵌套的和。具体来讲，假设 $\mathscr{M}$ 是一个语言的语素集，$\mathbf{M} \in \mathbb{R}^{n\times|\mathscr{M}|}$ 是语素嵌入矩阵,$\mathbf{m}^{\mathbf{j}}$ 是 $\mathbf{M}$ 的第 j 列，给定词 k, 下式为输入 LSTM 的词的表示：

$$\mathbf{x}^{\mathbf{k}} + \sum_{\mathbf{j}\in\mathscr{M}^{\mathbf{k}}} \mathbf{m}^{\mathbf{j}} \tag{16}$$

在这里 $\mathbf{x}^{\mathbf{k}}$ 是词嵌入，$\mathscr{M}_k \subset \mathscr{M}$ 是词 k 的词素集。词素通过在预处理阶段运行一个词素标记非监督算法来获得。词嵌入本身是在词素嵌入的基础上的。词素嵌入对于小模型和大模型的大小分别是 200/650。我们进一步训练词单位的 LSTM 作为另一个基准.

从表 4 中可以看出，我们的字单位的模型表现明显优于词单位的其他模型，尽管我们的模型更小。字模型同样优于词素模型（MLBL 和 LSTM 架构），注意到词素模型需要更多的参数，因为词嵌入作为输入的一部分。

由于内存限制，我们在 DATA-L 上只训练了小模型。有趣的是，在西班牙语、法语和

英语中，我们没有观察到词模型和语素模型显著的区别。字模型同样比词和语素取得更好多结果。在英语训练中，对于 $\mathcal{V}$ 很大大时候，我们同样也观察到复杂度显著的减少。作为这一小节的总结，我们要指出，对于不同语言，我们使用同样的架构，并且对于高阶参数，我们并没有针对特定的语言有特定的参数调整

		DATA-S					
		Cs	De	Es	Fr	Ru	Ar
Botha	KN-4	545	366	241	274	396	323
	MLBL	465	296	200	225	304	–
Small	Word	503	305	212	229	352	216
	Morph	414	278	197	216	290	230
	Char	401	260	182	189	278	196
Large	Word	493	286	200	222	357	172
	Morph	398	263	177	196	271	**148**
	Char	**371**	**239**	**165**	**184**	**261**	**148**

图 5 对于 DATA-S 在测试集上的复杂度

		DATA-L					
		Cs	De	Es	Fr	Ru	En
Botha	KN-4	862	463	219	243	390	291
	MLBL	643	404	203	227	**300**	273
Small	Word	701	347	186	202	353	236
	Morph	615	331	189	209	331	233
	Char	**578**	**305**	**169**	**190**	313	**216**

图 6 对于 DATA-L 在测试集上的复杂度

5 讨论

5.1 学习到的词表示

我们研究了模型在 PTB 数据集上学习到的词的表示。表 6 显示了词单位和字单位模型学习到的近邻词表示。对于字单位模型，我们比较了使用高速网络前后的

	In Vocabulary					Out-of-Vocabulary		
	while	*his*	*you*	*richard*	*trading*	*computer-aided*	*misinformed*	*looooook*
LSTM-Word	*although*	*your*	*conservatives*	*jonathan*	*advertised*	–	–	–
	letting	*her*	*we*	*robert*	*advertising*	–	–	–
	though	*my*	*guys*	*neil*	*turnover*	–	–	–
	minute	*their*	*i*	*nancy*	*turnover*	–	–	–
LSTM-Char (before highway)	*chile*	*this*	*your*	*hard*	*heading*	*computer-guided*	*informed*	*look*
	whole	*hhs*	*young*	*rich*	*training*	*computerized*	*performed*	*cook*
	meanwhile	*is*	*four*	*richer*	*reading*	*disk-drive*	*transformed*	*looks*
	white	*has*	*youth*	*richter*	*leading*	*computer*	*inform*	*shook*
LSTM-Char (after highway)	*meanwhile*	*hhs*	*we*	*eduard*	*trade*	*computer-guided*	*informed*	*look*
	whole	*this*	*your*	*gerard*	*training*	*computer-driven*	*performed*	*looks*
	though	*their*	*doug*	*edward*	*traded*	*computerized*	*outperformed*	*looked*
	nevertheless	*your*	*i*	*carl*	*trader*	*computer*	*transformed*	*looking*

图 7　最近邻词在字单位和词单位经过高速网络后的表达

词表示。

在进入高速网络层之前，词表示看起来只依赖于表面形式，例如单词 you 的最近邻是 your，young，four，youth。高速网络似乎能够编码和拼写无关的语义信息。在经过高速网络之后，you 的最近邻是 we，we 的拼写和 you 差很多。另一个例子是，while 和 though，他们看起来拼写很不一样，但是我们的模型能够把他们放置到临近的地方。我们的模型也犯了很多明显的错误（比如 his 和 hhs），尽管可能是由于数据集太小造成的，还是揭示了我们模型的一些局限。

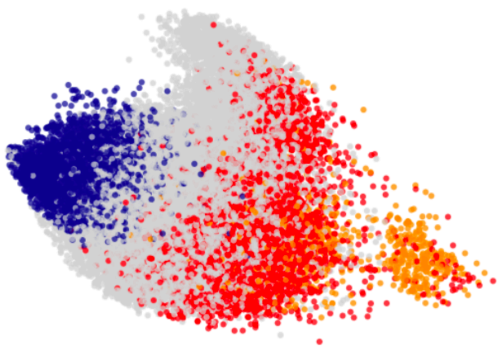

图 8　字的 n 元语法的点图

5.2 学习到的字 n 元语法表示

正如前面讨论到的，CharrCNN 的每个筛选器对于识别特定字的 n 元语法是必要的。我们最初的期待是每个筛选器能够学习在不同的语素上激活，然后根据识别出的语素实现语义表达。然而，通过研究筛选器选出的 n 元语法，我们发现它们并没有和合法的语素建立起联系。

为了能够建立起更好的关于字模型学了的什么的直觉，通过主成分分析，我们绘出了学习到的 n 元语法的表示。我们把每个字 n 元语法送入 CharCNN，然后利用 CharCNN 的输出作为对应的字的固定维度的 n 元语法表示。正如图表 2 显示的，我们的模型学习到了如何区分前缀（红）、后缀（蓝）和其他（灰）。我们也发现这个表达对于包含连字符的字 n 元语法表示特别敏感，我们的推测是因为连字符通常预示着这个词是演讲的一部分。

5.3 高速网络层

我们通过隔离分析定量的研究高速网络的作用。我们移除模型的一层高速网络层，发现模型的性能严重下降。因为性能的不同可能是由于模型大小减少所造成的，我们也训练了一个将 $\mathbf{y}^{k}$ 送入一个一层 MLP，然后将它作为 LSTM 的输入。尽管可能是因为优化的问题，但是我们发现 MLP 效果也很差。

我们推测高速网络对于 CNN 来说是必要的。CNN 已经在很多 NLP 任务中被证明是有效的方法。我们推测将高速网络结合进 CNN 中将会取得更好的结果。

我们也发现：（1）有一到两层高速网络是很重要的，但是进一步增加高速网络层数，并没有取得更优的结果。(2) 在最大池化前设置更多的卷积网络并没有帮助。（3）高速网络对于使用词嵌入作为输入的模型并没有帮助。

5.4 词汇量大小的影响

我们接着学习了词汇集的大小对于不同模型结果的影响。我们使用 DATA-L 中的德语（DE）数据集，实用不同的词汇集大小，计算从词单位和字单位复杂率减少

	LSTM-Char	
	Small	Large
No Highway Layers	100.3	84.6
One Highway Layer	92.3	79.7
Two Highway Layers	90.1	78.9
One MLP Layer	111.2	92.6

图 9　小模型和大模型在使用/不使用高速网络的复杂度对比

的大小。为了控制词汇量的变化，我们选出最常用的 k 个词，把他们替换为 <unk>。因为在前面的实验中，字模型没有使用 <unk> 的表面形式，只是把它当成另外一个标签。尽管表 8 表明随着词汇量的增加，复杂度减少的没有那么明显，我们还是发现字模型在所有的场景下都还是优于词模型。

		$\vert\mathcal{V}\vert$			
		10 k	25 k	50 k	100 k
T	1 m	17%	16%	21%	–
	5 m	8%	14%	16%	21%
	10 m	9%	9%	12%	15%
	25 m	9%	8%	9%	10%

图 10　不同的词汇量下，字模型比词模型在复杂度减少量

5.5　进一步观察

我们在进一步的的实验和观察中发现了下面的情况：

- 结合词嵌入和 CharCNN 的输出的词表示结果稍微糟糕一些。这个结果很让我们意外，因为有相关报道表明这样的结合在演讲标记和实体识别中有明显的改善。尽管我们的实验结果可能是由于我们的实验设置的不够合理，但是还是表明，对于一些任务，词嵌入式无用的，字输入就已经足够了

- 尽管我们的模型需要卷积操作，所以相对于只是进行简单词查找的词模型慢一些，但是我们还是发现这种速度的区别可以通过 GPU 实现的优化来控制的。比如，在 PTB 数据集上，大的字单位模型训练速度为 1500 标签每秒，相对应的，词模型训练速度为 3000 标签每秒。为了评分，因为 CharCNN 的输出可以被预先训练出来，所以我们的模型可以和词模型有相同的运行时间。然而，这样会增加模型的大小，所以这是一个在运行时速度和内存之间的取舍问题。

6 总结

我们引入了只使用字单位输入的神经语言模型。预测还是在词单位进行的，尽管模型有更少的参数，我们的模型表现超出使用词单位和使用语速单位的模型的基准线。我们的模型对于词嵌入作为输入在神经语言建模中的必要性。

对于从字模型中获取到的词的表示进行分析之后揭示了这个模型能够只从字编码丰富的语义信息和拼写信息。使用 CharCNN 和高速网络层对于表达学习还是有很大帮助的。

到目前为止，由于序列化处理词作为自然语言处理比较普世的方法，如果这篇论文的方法能够对其他任务有所帮助是很好的。

7 致谢

我们非常感谢 Jan Botha，他为我们提供了预处理好的数据和模型的结果。

Character-Aware Neural Language Models

Yoon Kim[†] Yacine Jernite[*] David Sontag[*] Alexander M. Rush[†]

[†]School of Engineering and Applied Sciences
Harvard University
{yoonkim,srush}@seas.harvard.edu

[*]Courant Institute of Mathematical Sciences
New York University
{jernite,dsontag}@cs.nyu.edu

Abstract

We describe a simple neural language model that relies only on character-level inputs. Predictions are still made at the word-level. Our model employs a convolutional neural network (CNN) and a highway network over characters, whose output is given to a long short-term memory (LSTM) recurrent neural network language model (RNN-LM). On the English Penn Treebank the model is on par with the existing state-of-the-art despite having 60% fewer parameters. On languages with rich morphology (Arabic, Czech, French, German, Spanish, Russian), the model outperforms word-level/morpheme-level LSTM baselines, again with fewer parameters. The results suggest that on many languages, character inputs are sufficient for language modeling. Analysis of word representations obtained from the character composition part of the model reveals that the model is able to encode, from characters only, both semantic and orthographic information.

Introduction

Language modeling is a fundamental task in artificial intelligence and natural language processing (NLP), with applications in speech recognition, text generation, and machine translation. A language model is formalized as a probability distribution over a sequence of strings (words), and traditional methods usually involve making an n-th order Markov assumption and estimating n-gram probabilities via counting and subsequent smoothing (Chen and Goodman 1998). The count-based models are simple to train, but probabilities of rare n-grams can be poorly estimated due to data sparsity (despite smoothing techniques).

Neural Language Models (NLM) address the n-gram data sparsity issue through parameterization of words as vectors (word embeddings) and using them as inputs to a neural network (Bengio, Ducharme, and Vincent 2003; Mikolov et al. 2010). The parameters are learned as part of the training process. Word embeddings obtained through NLMs exhibit the property whereby semantically close words are likewise close in the induced vector space (as is the case with non-neural techniques such as Latent Semantic Analysis (Deerwester, Dumais, and Harshman 1990)).

While NLMs have been shown to outperform count-based n-gram language models (Mikolov et al. 2011), they are blind to subword information (e.g. morphemes). For example, they do not know, a priori, that *eventful*, *eventfully*, *uneventful*, and *uneventfully* should have structurally related embeddings in the vector space. Embeddings of rare words can thus be poorly estimated, leading to high perplexities for rare words (and words surrounding them). This is especially problematic in morphologically rich languages with long-tailed frequency distributions or domains with dynamic vocabularies (e.g. social media).

In this work, we propose a language model that leverages subword information through a character-level convolutional neural network (CNN), whose output is used as an input to a recurrent neural network language model (RNN-LM). Unlike previous works that utilize subword information via morphemes (Botha and Blunsom 2014; Luong, Socher, and Manning 2013), our model does not require morphological tagging as a pre-processing step. And, unlike the recent line of work which combines input word embeddings with features from a character-level model (dos Santos and Zadrozny 2014; dos Santos and Guimaraes 2015), our model does not utilize word embeddings at all in the input layer. Given that most of the parameters in NLMs are from the word embeddings, the proposed model has significantly fewer parameters than previous NLMs, making it attractive for applications where model size may be an issue (e.g. cell phones).

To summarize, our contributions are as follows:

- on English, we achieve results on par with the existing state-of-the-art on the Penn Treebank (PTB), despite having approximately 60% fewer parameters, and
- on morphologically rich languages (Arabic, Czech, French, German, Spanish, and Russian), our model outperforms various baselines (Kneser-Ney, word-level/morpheme-level LSTM), again with fewer parameters.

We have released all the code for the models described in this paper.[1]

[1]https://github.com/yoonkim/lstm-char-cnn

Model

The architecture of our model, shown in Figure 1, is straightforward. Whereas a conventional NLM takes word embeddings as inputs, our model instead takes the output from a single-layer character-level convolutional neural network with max-over-time pooling.

For notation, we denote vectors with bold lower-case (e.g. $\mathbf{x}_t, \mathbf{b}$), matrices with bold upper-case (e.g. $\mathbf{W}, \mathbf{U}^o$), scalars with italic lower-case (e.g. x, b), and sets with cursive upper-case (e.g. $\mathcal{V}, \mathcal{C}$) letters. For notational convenience we assume that words and characters have already been converted into indices.

Recurrent Neural Network

A recurrent neural network (RNN) is a type of neural network architecture particularly suited for modeling sequential phenomena. At each time step t, an RNN takes the input vector $\mathbf{x}_t \in \mathbb{R}^n$ and the hidden state vector $\mathbf{h}_{t-1} \in \mathbb{R}^m$ and produces the next hidden state $\mathbf{h}_t$ by applying the following recursive operation:

$$\mathbf{h}_t = f(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}) \tag{1}$$

Here $\mathbf{W} \in \mathbb{R}^{m \times n}, \mathbf{U} \in \mathbb{R}^{m \times m}, \mathbf{b} \in \mathbb{R}^m$ are parameters of an affine transformation and f is an element-wise nonlinearity. In theory the RNN can summarize all historical information up to time t with the hidden state $\mathbf{h}_t$. In practice however, learning long-range dependencies with a vanilla RNN is difficult due to vanishing/exploding gradients (Bengio, Simard, and Frasconi 1994), which occurs as a result of the Jacobian's multiplicativity with respect to time.

Long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) addresses the problem of learning long range dependencies by augmenting the RNN with a memory cell vector $\mathbf{c}_t \in \mathbb{R}^n$ at each time step. Concretely, one step of an LSTM takes as input $\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}$ and produces $\mathbf{h}_t$, $\mathbf{c}_t$ via the following intermediate calculations:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}^i\mathbf{x}_t + \mathbf{U}^i\mathbf{h}_{t-1} + \mathbf{b}^i)\\
\mathbf{f}_t &= \sigma(\mathbf{W}^f\mathbf{x}_t + \mathbf{U}^f\mathbf{h}_{t-1} + \mathbf{b}^f)\\
\mathbf{o}_t &= \sigma(\mathbf{W}^o\mathbf{x}_t + \mathbf{U}^o\mathbf{h}_{t-1} + \mathbf{b}^o)\\
\mathbf{g}_t &= \tanh(\mathbf{W}^g\mathbf{x}_t + \mathbf{U}^g\mathbf{h}_{t-1} + \mathbf{b}^g)\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned} \tag{2}$$

Here $\sigma(\cdot)$ and $\tanh(\cdot)$ are the element-wise sigmoid and hyperbolic tangent functions, $\odot$ is the element-wise multiplication operator, and $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{o}_t$ are referred to as *input*, *forget*, and *output* gates. At $t = 1$, $\mathbf{h}_0$ and $\mathbf{c}_0$ are initialized to zero vectors. Parameters of the LSTM are $\mathbf{W}^j, \mathbf{U}^j, \mathbf{b}^j$ for $j \in \{i, f, o, g\}$.

Memory cells in the LSTM are additive with respect to time, alleviating the gradient vanishing problem. Gradient exploding is still an issue, though in practice simple optimization strategies (such as gradient clipping) work well. LSTMs have been shown to outperform vanilla RNNs on many tasks, including on language modeling (Sundermeyer, Schluter, and Ney 2012). It is easy to extend the RNN/LSTM to two (or more) layers by having another network whose input at t is $\mathbf{h}_t$ (from the first network). Indeed, having multiple layers is often crucial for obtaining competitive performance on various tasks (Pascanu et al. 2013).

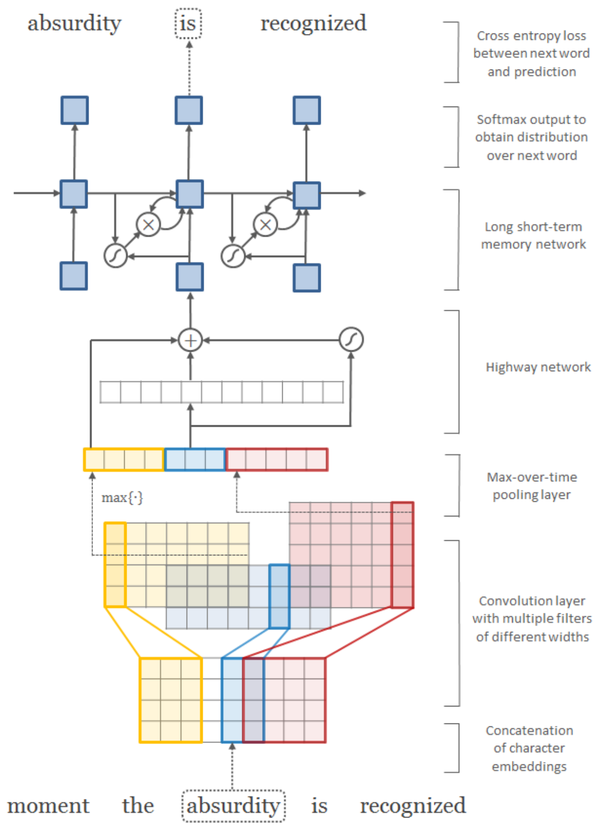
absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized

Figure 1: Architecture of our language model applied to an example sentence. Best viewed in color. Here the model takes *absurdity* as the current input and combines it with the history (as represented by the hidden state) to predict the next word, *is*. First layer performs a lookup of character embeddings (of dimension four) and stacks them to form the matrix $\mathbf{C}^k$. Then convolution operations are applied between $\mathbf{C}^k$ and multiple filter matrices. Note that in the above example we have twelve filters—three filters of width two (blue), four filters of width three (yellow), and five filters of width four (red). A max-over-time pooling operation is applied to obtain a fixed-dimensional representation of the word, which is given to the highway network. The highway network's output is used as the input to a multi-layer LSTM. Finally, an affine transformation followed by a softmax is applied over the hidden representation of the LSTM to obtain the distribution over the next word. Cross entropy loss between the (predicted) distribution over next word and the actual next word is minimized. Element-wise addition, multiplication, and sigmoid operators are depicted in circles, and affine transformations (plus nonlinearities where appropriate) are represented by solid arrows.

Recurrent Neural Network Language Model

Let $\mathcal{V}$ be the fixed size vocabulary of words. A language model specifies a distribution over w_{t+1} (whose support is $\mathcal{V}$) given the historical sequence $w_{1:t} = [w_1, \ldots, w_t]$. A recurrent neural network language model (RNN-LM) does this by applying an affine transformation to the hidden layer followed by a softmax:

$$\Pr(w_{t+1} = j | w_{1:t}) = \frac{\exp(\mathbf{h}_t \cdot \mathbf{p}^j + q^j)}{\sum_{j' \in \mathcal{V}} \exp(\mathbf{h}_t \cdot \mathbf{p}^{j'} + q^{j'})} \tag{3}$$

where $\mathbf{p}^j$ is the j-th column of $\mathbf{P} \in \mathbb{R}^{m \times |\mathcal{V}|}$ (also referred to as the *output embedding*),[2] and q^j is a bias term. Similarly, for a conventional RNN-LM which usually takes words as inputs, if $w_t = k$, then the input to the RNN-LM at t is the *input embedding* $\mathbf{x}^k$, the k-th column of the embedding matrix $\mathbf{X} \in \mathbb{R}^{n \times |\mathcal{V}|}$. Our model simply replaces the input embeddings $\mathbf{X}$ with the output from a character-level convolutional neural network, to be described below.

If we denote $w_{1:T} = [w_1, \cdots, w_T]$ to be the sequence of words in the training corpus, training involves minimizing the negative log-likelihood (NLL) of the sequence

$$NLL = -\sum_{t=1}^{T} \log \Pr(w_t | w_{1:t-1}) \tag{4}$$

which is typically done by truncated backpropagation through time (Werbos 1990; Graves 2013).

Character-level Convolutional Neural Network

In our model, the input at time t is an output from a character-level convolutional neural network (CharCNN), which we describe in this section. CNNs (LeCun et al. 1989) have achieved state-of-the-art results on computer vision (Krizhevsky, Sutskever, and Hinton 2012) and have also been shown to be effective for various NLP tasks (Collobert et al. 2011). Architectures employed for NLP applications differ in that they typically involve temporal rather than spatial convolutions.

Let $\mathcal{C}$ be the vocabulary of characters, d be the dimensionality of character embeddings,[3] and $\mathbf{Q} \in \mathbb{R}^{d \times |\mathcal{C}|}$ be the matrix character embeddings. Suppose that word $k \in \mathcal{V}$ is made up of a sequence of characters $[c_1, \ldots, c_l]$, where l is the length of word k. Then the character-level representation of k is given by the matrix $\mathbf{C}^k \in \mathbb{R}^{d \times l}$, where the j-th column corresponds to the character embedding for c_j (i.e. the c_j-th column of $\mathbf{Q}$).[4]

We apply a narrow convolution between $\mathbf{C}^k$ and a *filter* (or *kernel*) $\mathbf{H} \in \mathbb{R}^{d \times w}$ of width w, after which we add a bias and apply a nonlinearity to obtain a *feature map* $\mathbf{f}^k \in \mathbb{R}^{l-w+1}$. Specifically, the i-th element of $\mathbf{f}^k$ is given by:

$$\mathbf{f}^k[i] = \tanh(\langle \mathbf{C}^k[*, i : i + w - 1], \mathbf{H} \rangle + b) \tag{5}$$

where $\mathbf{C}^k[*, i : i+w-1]$ is the i-to-$(i+w-1)$-th column of $\mathbf{C}^k$ and $\langle \mathbf{A}, \mathbf{B} \rangle = \mathrm{Tr}(\mathbf{A}\mathbf{B}^T)$ is the Frobenius inner product. Finally, we take the *max-over-time*

$$y^k = \max_i \mathbf{f}^k[i] \tag{6}$$

as the feature corresponding to the filter $\mathbf{H}$ (when applied to word k). The idea is to capture the most important feature—the one with the highest value—for a given filter. A filter is essentially picking out a character n-gram, where the size of the n-gram corresponds to the filter width.

We have described the process by which *one* feature is obtained from *one* filter matrix. Our CharCNN uses multiple filters of varying widths to obtain the feature vector for k. So if we have a total of h filters $\mathbf{H}_1, \ldots, \mathbf{H}_h$, then $\mathbf{y}^k = [y_1^k, \ldots, y_h^k]$ is the input representation of k. For many NLP applications h is typically chosen to be in $[100, 1000]$.

Highway Network

We could simply replace $\mathbf{x}^k$ (the word embedding) with $\mathbf{y}^k$ at each t in the RNN-LM, and as we show later, this simple model performs well on its own (Table 7). One could also have a multilayer perceptron (MLP) over $\mathbf{y}^k$ to model interactions between the character n-grams picked up by the filters, but we found that this resulted in worse performance.

Instead we obtained improvements by running $\mathbf{y}^k$ through a *highway network*, recently proposed by Srivastava et al. (2015). Whereas one layer of an MLP applies an affine transformation followed by a nonlinearity to obtain a new set of features,

$$\mathbf{z} = g(\mathbf{W}\mathbf{y} + \mathbf{b}) \tag{7}$$

one layer of a highway network does the following:

$$\mathbf{z} = \mathbf{t} \odot g(\mathbf{W}_H \mathbf{y} + \mathbf{b}_H) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{y} \tag{8}$$

where g is a nonlinearity, $\mathbf{t} = \sigma(\mathbf{W}_T \mathbf{y} + \mathbf{b}_T)$ is called the *transform* gate, and $(\mathbf{1} - \mathbf{t})$ is called the *carry* gate. Similar to the memory cells in LSTM networks, highway layers allow for training of deep networks by adaptively *carrying* some dimensions of the input directly to the output.[5] By construction the dimensions of $\mathbf{y}$ and $\mathbf{z}$ have to match, and hence $\mathbf{W}_T$ and $\mathbf{W}_H$ are square matrices.

Experimental Setup

As is standard in language modeling, we use perplexity (PPL) to evaluate the performance of our models. Perplexity of a model over a sequence $[w_1, \ldots, w_T]$ is given by

$$PPL = \exp\Big(\frac{NLL}{T}\Big) \tag{9}$$

where NLL is calculated over the test set. We test the model on corpora of varying languages and sizes (statistics available in Table 1).

We conduct hyperparameter search, model introspection, and ablation studies on the English Penn Treebank (PTB) (Marcus, Santorini, and Marcinkiewicz 1993), utilizing the

[2]In our work, predictions are at the word-level, and hence we still utilize word embeddings in the output layer.

[3]Given that $|\mathcal{C}|$ is usually small, some authors work with one-hot representations of characters. However we found that using lower dimensional representations of characters (i.e. $d < |\mathcal{C}|$) performed slightly better.

[4]Two technical details warrant mention here: (1) we append start-of-word and end-of-word characters to each word to better represent prefixes and suffixes and hence $\mathbf{C}^k$ actually has $l + 2$ columns; (2) for batch processing, we zero-pad $\mathbf{C}^k$ so that the number of columns is constant (equal to the max word length) for all words in $\mathcal{V}$.

[5]Srivastava et al. (2015) recommend initializing $\mathbf{b}_T$ to a negative value, in order to militate the initial behavior towards *carry*. We initialized $\mathbf{b}_T$ to a small interval around -2.

	DATA-S			DATA-L		
	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{C}\rvert$	T	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{C}\rvert$	T
English (EN)	10 k	51	1 m	60 k	197	20 m
Czech (CS)	46 k	101	1 m	206 k	195	17 m
German (DE)	37 k	74	1 m	339 k	260	51 m
Spanish (ES)	27 k	72	1 m	152 k	222	56 m
French (FR)	25 k	76	1 m	137 k	225	57 m
Russian (RU)	62 k	62	1 m	497 k	111	25 m
Arabic (AR)	86 k	132	4 m	–	–	–

Table 1: Corpus statistics. $|\mathcal{V}|$ = word vocabulary size; $|\mathcal{C}|$ = character vocabulary size; T = number of tokens in training set. The small English data is from the Penn Treebank and the Arabic data is from the News-Commentary corpus. The rest are from the 2013 ACL Workshop on Machine Translation. $|\mathcal{C}|$ is large because of (rarely occurring) special characters.

		Small	Large
CNN	d	15	15
	w	$[1, 2, 3, 4, 5, 6]$	$[1, 2, 3, 4, 5, 6, 7]$
	h	$[25 \cdot w]$	$[\min\{200, 50 \cdot w\}]$
	f	tanh	tanh
Highway	l	1	2
	g	ReLU	ReLU
LSTM	l	2	2
	m	300	650

Table 2: Architecture of the small and large models. d = dimensionality of character embeddings; w = filter widths; h = number of filter matrices, as a function of filter width (so the large model has filters of width $[1, 2, 3, 4, 5, 6, 7]$ of size $[50, 100, 150, 200, 200, 200, 200]$ for a total of 1100 filters); f, g = nonlinearity functions; l = number of layers; m = number of hidden units.

standard training (0-20), validation (21-22), and test (23-24) splits along with pre-processing by Mikolov et al. (2010). With approximately 1m tokens and $|\mathcal{V}| = 10$k, this version has been extensively used by the language modeling community and is publicly available.[6]

With the optimal hyperparameters tuned on PTB, we apply the model to various morphologically rich languages: Czech, German, French, Spanish, Russian, and Arabic. Non-Arabic data comes from the 2013 ACL Workshop on Machine Translation,[7] and we use the same train/validation/test splits as in Botha and Blunsom (2014). While the raw data are publicly available, we obtained the preprocessed versions from the authors,[8] whose morphological NLM serves as a baseline for our work. We train on both the small datasets (DATA-S) with 1m tokens per language, and the large datasets (DATA-L) including the large English data which has a much bigger $|\mathcal{V}|$ than the PTB. Arabic data comes from the News-Commentary corpus,[9] and we perform our own preprocessing and train/validation/test splits.

In these datasets only singleton words were replaced with <unk> and hence we effectively use the full vocabulary. It is worth noting that the character model can utilize surface forms of OOV tokens (which were replaced with <unk>), but we do not do this and stick to the preprocessed versions (despite disadvantaging the character models) for exact comparison against prior work.

Optimization

The models are trained by truncated backpropagation through time (Werbos 1990; Graves 2013). We backpropagate for 35 time steps using stochastic gradient descent where the learning rate is initially set to 1.0 and halved if the perplexity does not decrease by more than 1.0 on the validation set after an epoch. On DATA-S we use a batch size of 20 and on DATA-L we use a batch size of 100 (for greater efficiency). Gradients are averaged over each batch. We train for 25 epochs on non-Arabic and 30 epochs on Arabic data (which was sufficient for convergence), picking the best performing model on the validation set. Parameters of the model are randomly initialized over a uniform distribution with support $[-0.05, 0.05]$.

For regularization we use dropout (Hinton et al. 2012) with probability 0.5 on the LSTM input-to-hidden layers (except on the initial Highway to LSTM layer) and the hidden-to-output softmax layer. We further constrain the norm of the gradients to be below 5, so that if the L_2 norm of the gradient exceeds 5 then we renormalize it to have $||\cdot|| = 5$ before updating. The gradient norm constraint was crucial in training the model. These choices were largely guided by previous work of Zaremba et al. (2014) on word-level language modeling with LSTMs.

Finally, in order to speed up training on DATA-L we employ a hierarchical softmax (Morin and Bengio 2005)—a common strategy for training language models with very large $|\mathcal{V}|$—instead of the usual softmax. We pick the number of clusters $c = \lceil\sqrt{|\mathcal{V}|}\rceil$ and randomly split $\mathcal{V}$ into mutually exclusive and collectively exhaustive subsets $\mathcal{V}_1, \dots, \mathcal{V}_c$ of (approximately) equal size.[10] Then $\Pr(w_{t+1} = j|w_{1:t})$ becomes,

$$\begin{aligned}\Pr(w_{t+1} = j|w_{1:t}) = &\frac{\exp(\mathbf{h}_t \cdot \mathbf{s}^r + t^r)}{\sum_{r'=1}^{c} \exp(\mathbf{h}_t \cdot \mathbf{s}^{r'} + t^{r'})} \\ &\times \frac{\exp(\mathbf{h}_t \cdot \mathbf{p}_r^j + q_r^j)}{\sum_{j' \in \mathcal{V}_r} \exp(\mathbf{h}_t \cdot \mathbf{p}_r^{j'} + q_r^{j'})}\end{aligned} \tag{10}$$

where r is the cluster index such that $j \in \mathcal{V}_r$. The first term is simply the probability of picking cluster r, and the second

[6]http://www.fit.vutbr.cz/~imikolov/rnnlm/

[7]http://www.statmt.org/wmt13/translation-task.html

[8]http://bothameister.github.io/

[9]http://opus.lingfil.uu.se/News-Commentary.php

[10]While Brown clustering/frequency-based clustering is commonly used in the literature (e.g. Botha and Blunsom (2014) use Brown clusering), we used random clusters as our implementation enjoys the best speed-up when the number of words in each cluster is approximately equal. We found random clustering to work surprisingly well.

	PPL	Size
LSTM-Word-Small	97.6	5 m
LSTM-Char-Small	92.3	5 m
LSTM-Word-Large	85.4	20 m
LSTM-Char-Large	78.9	19 m
KN-5 (Mikolov et al. 2012)	141.2	2 m
RNN[†] (Mikolov et al. 2012)	124.7	6 m
RNN-LDA[†] (Mikolov et al. 2012)	113.7	7 m
genCNN[†] (Wang et al. 2015)	116.4	8 m
FOFE-FNNLM[†] (Zhang et al. 2015)	108.0	6 m
Deep RNN (Pascanu et al. 2013)	107.5	6 m
Sum-Prod Net[†] (Cheng et al. 2014)	100.0	5 m
LSTM-1[†] (Zaremba et al. 2014)	82.7	20 m
LSTM-2[†] (Zaremba et al. 2014)	78.4	52 m

Table 3: Performance of our model versus other neural language models on the English Penn Treebank test set. PPL refers to perplexity (lower is better) and size refers to the approximate number of parameters in the model. KN-5 is a Kneser-Ney 5-gram language model which serves as a non-neural baseline. [†]For these models the authors did not explicitly state the number of parameters, and hence sizes shown here are estimates based on our understanding of their papers or private correspondence with the respective authors.

		DATA-S					
		CS	DE	ES	FR	RU	AR
Botha	KN-4	545	366	241	274	396	323
	MLBL	465	296	200	225	304	–
Small	Word	503	305	212	229	352	216
	Morph	414	278	197	216	290	230
	Char	401	260	182	189	278	196
Large	Word	493	286	200	222	357	172
	Morph	398	263	177	196	271	**148**
	Char	**371**	**239**	**165**	**184**	**261**	**148**

Table 4: Test set perplexities for DATA-S. First two rows are from Botha (2014) (except on Arabic where we trained our own KN-4 model) while the last six are from this paper. KN-4 is a Kneser-Ney 4-gram language model, and MLBL is the best performing morphological logbilinear model from Botha (2014). Small/Large refer to model size (see Table 2), and Word/Morph/Char are models with words/morphemes/characters as inputs respectively.

term is the probability of picking word j given that cluster r is picked. We found that hierarchical softmax was not necessary for models trained on DATA-S.

Results

English Penn Treebank

We train two versions of our model to assess the trade-off between performance and size. Architecture of the small (LSTM-Char-Small) and large (LSTM-Char-Large) models is summarized in Table 2. As another baseline, we also train two comparable LSTM models that use word embeddings only (LSTM-Word-Small, LSTM-Word-Large). LSTM-Word-Small uses 200 hidden units and LSTM-Word-Large uses 650 hidden units. Word embedding sizes are also 200 and 650 respectively. These were chosen to keep the number of parameters similar to the corresponding character-level model.

As can be seen from Table 3, our large model is on par with the existing state-of-the-art (Zaremba et al. 2014), despite having approximately 60% fewer parameters. Our small model significantly outperforms other NLMs of similar size, even though it is penalized by the fact that the dataset already has OOV words replaced with <unk> (other models are purely word-level models). While lower perplexities have been reported with model ensembles (Mikolov and Zweig 2012), we do not include them here as they are not comparable to the current work.

Other Languages

The model's performance on the English PTB is informative to the extent that it facilitates comparison against the large body of existing work. However, English is relatively simple from a morphological standpoint, and thus our next set of results (and arguably the main contribution of this paper) is focused on languages with richer morphology (Table 4, Table 5).

We compare our results against the morphological log-bilinear (MLBL) model from Botha and Blunsom (2014), whose model also takes into account subword information through morpheme embeddings that are summed at the input and output layers. As comparison against the MLBL models is confounded by our use of LSTMs—widely known to outperform their feed-forward/log-bilinear cousins—we also train an LSTM version of the morphological NLM, where the input representation of a word given to the LSTM is a summation of the word's morpheme embeddings. Concretely, suppose that $\mathcal{M}$ is the set of morphemes in a language, $\mathbf{M} \in \mathbb{R}^{n\times|\mathcal{M}|}$ is the matrix of morpheme embeddings, and $\mathbf{m}^j$ is the j-th column of $\mathbf{M}$ (i.e. a morpheme embedding). Given the input word k, we feed the following representation to the LSTM:

$$\mathbf{x}^k + \sum_{j\in\mathcal{M}_k} \mathbf{m}^j \tag{11}$$

where $\mathbf{x}^k$ is the word embedding (as in a word-level model) and $\mathcal{M}_k \subset \mathcal{M}$ is the set of morphemes for word k. The morphemes are obtained by running an unsupervised morphological tagger as a preprocessing step.[11] We emphasize that the word embedding itself (i.e. $\mathbf{x}^k$) is added on top of the morpheme embeddings, as was done in Botha and Blunsom (2014). The morpheme embeddings are of size 200/650 for the small/large models respectively. We further train word-level LSTM models as another baseline.

On DATA-S it is clear from Table 4 that the character-level models outperform their word-level counterparts de-

[11]We use *Morfessor Cat-MAP* (Creutz and Lagus 2007), as in Botha and Blunsom (2014).

		DATA-L					
		CS	DE	ES	FR	RU	EN
Botha	KN-4	862	463	219	243	390	291
	MLBL	643	404	203	227	**300**	273
Small	Word	701	347	186	202	353	236
	Morph	615	331	189	209	331	233
	Char	**578**	**305**	**169**	**190**	313	**216**

Table 5: Test set perplexities on DATA-L. First two rows are from Botha (2014), while the last three rows are from the small LSTM models described in the paper. KN-4 is a Kneser-Ney 4-gram language model, and MLBL is the best performing morphological log-bilinear model from Botha (2014). Word/Morph/Char are models with words/morphemes/characters as inputs respectively.

spite, again, being smaller.[12] The character models also outperform their morphological counterparts (both MLBL and LSTM architectures), although improvements over the morphological LSTMs are more measured. Note that the morpheme models have strictly more parameters than the word models because word embeddings are used as part of the input.

Due to memory constraints[13] we only train the small models on DATA-L (Table 5). Interestingly we do not observe significant differences going from word to morpheme LSTMs on Spanish, French, and English. The character models again outperform the word/morpheme models. We also observe significant perplexity reductions even on English when $\mathcal{V}$ is large. We conclude this section by noting that we used the same architecture for all languages and did not perform any language-specific tuning of hyperparameters.

Discussion

Learned Word Representations

We explore the word representations learned by the models on the PTB. Table 6 has the nearest neighbors of word representations learned from both the word-level and character-level models. For the character models we compare the representations obtained before and after highway layers.

Before the highway layers the representations seem to solely rely on surface forms—for example the nearest neighbors of *you* are *your, young, four, youth*, which are close to *you* in terms of edit distance. The highway layers however, seem to enable encoding of semantic features that are not discernable from orthography alone. After highway layers the nearest neighbor of *you* is *we*, which is orthographically distinct from *you*. Another example is *while* and *though*—these words are far apart edit distance-wise yet the composition model is able to place them near each other. The model

[12]The difference in parameters is greater for non-PTB corpora as the size of the word model scales faster with $|\mathcal{V}|$. For example, on Arabic the small/large word models have 35m/121m parameters while the corresponding character models have 29m/69m parameters respectively.

[13]All models were trained on GPUs with 2GB memory.

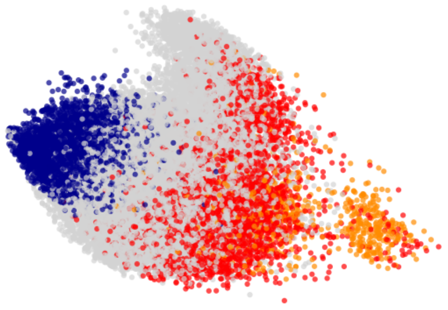

Figure 2: Plot of character n-gram representations via PCA for English. Colors correspond to: prefixes (red), suffixes (blue), hyphenated (orange), and all others (grey). Prefixes refer to character n-grams which start with the start-of-word character. Suffixes likewise refer to character n-grams which end with the end-of-word character.

also makes some clear mistakes (e.g. *his* and *hhs*), highlighting the limits of our approach, although this could be due to the small dataset.

The learned representations of OOV words (*computer-aided*, *misinformed*) are positioned near words with the same part-of-speech. The model is also able to correct for incorrect/non-standard spelling (*looooook*), indicating potential applications for text normalization in noisy domains.

Learned Character N-gram Representations

As discussed previously, each filter of the CharCNN is essentially learning to detect particular character n-grams. Our initial expectation was that each filter would learn to activate on different morphemes and then build up semantic representations of words from the identified morphemes. However, upon reviewing the character n-grams picked up by the filters (i.e. those that maximized the value of the filter), we found that they did not (in general) correspond to valid morphemes.

To get a better intuition for what the character composition model is learning, we plot the learned representations of all character n-grams (that occurred as part of at least two words in $\mathcal{V}$) via principal components analysis (Figure 2). We feed each character n-gram into the CharCNN and use the CharCNN's output as the fixed dimensional representation for the corresponding character n-gram. As is apparent from Figure 2, the model learns to differentiate between prefixes (red), suffixes (blue), and others (grey). We also find that the representations are particularly sensitive to character n-grams containing hyphens (orange), presumably because this is a strong signal of a word's part-of-speech.

Highway Layers

We quantitatively investigate the effect of highway network layers via ablation studies (Table 7). We train a model without any highway layers, and find that performance decreases significantly. As the difference in performance could be due to the decrease in model size, we also train a model that feeds $\mathbf{y}^k$ (i.e. word representation from the CharCNN)

	In Vocabulary					Out-of-Vocabulary		
	while	*his*	*you*	*richard*	*trading*	*computer-aided*	*misinformed*	*looooook*
LSTM-Word	*although*	*your*	*conservatives*	*jonathan*	*advertised*	–	–	–
	letting	*her*	*we*	*robert*	*advertising*	–	–	–
	though	*my*	*guys*	*neil*	*turnover*	–	–	–
	minute	*their*	*i*	*nancy*	*turnover*	–	–	–
LSTM-Char (before highway)	*chile*	*this*	*your*	*hard*	*heading*	*computer-guided*	*informed*	*look*
	whole	*hhs*	*young*	*rich*	*training*	*computerized*	*performed*	*cook*
	meanwhile	*is*	*four*	*richer*	*reading*	*disk-drive*	*transformed*	*looks*
	white	*has*	*youth*	*richter*	*leading*	*computer*	*inform*	*shook*
LSTM-Char (after highway)	*meanwhile*	*hhs*	*we*	*eduard*	*trade*	*computer-guided*	*informed*	*look*
	whole	*this*	*your*	*gerard*	*training*	*computer-driven*	*performed*	*looks*
	though	*their*	*doug*	*edward*	*traded*	*computerized*	*outperformed*	*looked*
	nevertheless	*your*	*i*	*carl*	*trader*	*computer*	*transformed*	*looking*

Table 6: Nearest neighbor words (based on cosine similarity) of word representations from the large word-level and character-level (before and after highway layers) models trained on the PTB. Last three words are OOV words, and therefore they do not have representations in the word-level model.

	LSTM-Char	
	Small	Large
No Highway Layers	100.3	84.6
One Highway Layer	92.3	79.7
Two Highway Layers	90.1	78.9
One MLP Layer	111.2	92.6

Table 7: Perplexity on the Penn Treebank for small/large models trained with/without highway layers.

through a one-layer multilayer perceptron (MLP) to use as input into the LSTM. We find that the MLP does poorly, although this could be due to optimization issues.

We hypothesize that highway networks are especially well-suited to work with CNNs, adaptively combining local features detected by the individual filters. CNNs have already proven to be been successful for many NLP tasks (Collobert et al. 2011; Shen et al. 2014; Kalchbrenner, Grefenstette, and Blunsom 2014; Kim 2014; Zhang, Zhao, and LeCun 2015; Lei, Barzilay, and Jaakola 2015), and we posit that further gains could be achieved by employing highway layers on top of existing CNN architectures.

We also anecdotally note that (1) having one to two highway layers was important, but more highway layers generally resulted in similar performance (though this may depend on the size of the datasets), (2) having more convolutional layers before max-pooling did not help, and (3) highway layers did not improve models that only used word embeddings as inputs.

Effect of Corpus/Vocab Sizes

We next study the effect of training corpus/vocabulary sizes on the relative performance between the different models. We take the German (DE) dataset from DATA-L and vary the training corpus/vocabulary sizes, calculating the perplexity reductions as a result of going from a small word-level model to a small character-level model. To vary the vocabulary size we take the most frequent k words and replace the rest with <unk>. As with previous experiments the character model does not utilize surface forms of <unk> and simply treats it as another token. Although Table 8 suggests that the perplexity reductions become less pronounced as the corpus size increases, we nonetheless find that the character-level model outperforms the word-level model in all scenarios.

		$\lvert\mathcal{V}\rvert$			
		10 k	25 k	50 k	100 k
T	1 m	17%	16%	21%	–
	5 m	8%	14%	16%	21%
	10 m	9%	9%	12%	15%
	25 m	9%	8%	9%	10%

Table 8: Perplexity reductions by going from small word-level to character-level models based on different corpus/vocabulary sizes on German (DE). $\lvert\mathcal{V}\rvert$ is the vocabulary size and T is the number of tokens in the training set. The full vocabulary of the 1m dataset was less than 100k and hence that scenario is unavailable.

Further Observations

We report on some further experiments and observations:

- Combining word embeddings with the CharCNN's output to form a combined representation of a word (to be used as input to the LSTM) resulted in slightly worse performance (81 on PTB with a large model). This was surprising, as improvements have been reported on part-of-speech tagging (dos Santos and Zadrozny 2014) and named entity recognition (dos Santos and Guimaraes 2015) by concatenating word embeddings with the output from a character-level CNN. While this could be due to insufficient experimentation on our part,[14] it suggests that for some tasks, word embeddings are superfluous—character inputs are good enough.

- While our model requires additional convolution operations over characters and is thus slower than a comparable word-level model which can perform a simple lookup at the input layer, we found that the difference was manageable with optimized GPU implementations—for example on PTB the large character-level model trained at 1500 tokens/sec compared to the word-level model which trained at 3000 tokens/sec. For scoring, our model can have the same running time as a pure word-level model, as the CharCNN's outputs can be pre-computed for all words in $\mathcal{V}$. This would, however, be at the expense of increased model size, and thus a trade-off can be made between run-time speed and memory (e.g. one could restrict the pre-computation to the most frequent words).

Related Work

Neural Language Models (NLM) encompass a rich family of neural network architectures for language modeling. Some example architectures include feed-forward (Bengio, Ducharme, and Vincent 2003), recurrent (Mikolov et al. 2010), sum-product (Cheng et al. 2014), log-bilinear (Mnih and Hinton 2007), and convolutional (Wang et al. 2015) networks.

In order to address the rare word problem, Alexandrescu and Kirchhoff (2006)—building on analogous work on count-based n-gram language models by Bilmes and Kirchhoff (2003)—represent a word as a set of shared factor embeddings. Their Factored Neural Language Model (FNLM) can incorporate morphemes, word shape information (e.g. capitalization) or any other annotation (e.g. part-of-speech tags) to represent words.

A specific class of FNLMs leverages morphemic information by viewing a word as a function of its (learned) morpheme embeddings (Luong, Socher, and Manning 2013; Botha and Blunsom 2014; Qui et al. 2014). For example Luong, Socher, and Manning (2013) apply a recursive neural network over morpheme embeddings to obtain the embedding for a single word. While such models have proved useful, they require morphological tagging as a preprocessing step.

Another direction of work has involved purely character-level NLMs, wherein both input and output are characters (Sutskever, Martens, and Hinton 2011; Graves 2013). Character-level models obviate the need for morphological tagging or manual feature engineering, and have the attractive property of being able to generate novel words. However they are generally outperformed by word-level models (Mikolov et al. 2012).

Outside of language modeling, improvements have been reported on part-of-speech tagging (dos Santos and Zadrozny 2014) and named entity recognition (dos Santos and Guimaraes 2015) by representing a word as a concatenation of its word embedding and an output from a character-level CNN, and using the combined representation as features in a Conditional Random Field (CRF). Zhang, Zhao, and LeCun (2015) do away with word embeddings completely and show that for text classification, a deep CNN over characters performs well. Ballesteros, Dyer, and Smith (2015) use an RNN over characters only to train a transition-based parser, obtaining improvements on many morphologically rich languages.

Finally, Ling et al. (2015) apply a bi-directional LSTM over characters to use as inputs for language modeling and part-of-speech tagging. They show improvements on various languages (English, Portuguese, Catalan, German, Turkish). It remains open as to which character composition model (i.e. CNN or LSTM) performs better.

[14]We experimented with (1) concatenation, (2) tensor products, (3) averaging, and (4) adaptive weighting schemes whereby the model learns a convex combination of word embeddings and the CharCNN outputs.

Conclusion

We have introduced a neural language model that utilizes only character-level inputs. Predictions are still made at the word-level. Despite having fewer parameters, our model outperforms baseline models that utilize word/morpheme embeddings in the input layer. Our work questions the necessity of word embeddings (as inputs) for neural language modeling.

Analysis of word representations obtained from the character composition part of the model further indicates that the model is able to encode, from characters only, rich semantic and orthographic features. Using the CharCNN and highway layers for representation learning (e.g. as input into word2vec (Mikolov et al. 2013)) remains an avenue for future work.

Insofar as sequential processing of words as inputs is ubiquitous in natural language processing, it would be interesting to see if the architecture introduced in this paper is viable for other tasks—for example, as an encoder/decoder in neural machine translation (Cho et al. 2014; Sutskever, Vinyals, and Le 2014).

Acknowledgments

We are especially grateful to Jan Botha for providing the preprocessed datasets and the model results.

References

Alexandrescu, A., and Kirchhoff, K. 2006. Factored Neural Language Models. In *Proceedings of NAACL.*

Ballesteros, M.; Dyer, C.; and Smith, N. A. 2015. Improved Transition-Based Parsing by Modeling Characters instead of Words with LSTMs. In *Proceedings of EMNLP.*

Bengio, Y.; Ducharme, R.; and Vincent, P. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research* 3:1137–1155.

Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning Long-term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks* 5:157–166.

Bilmes, J., and Kirchhoff, K. 2003. Factored Language Models and Generalized Parallel Backoff. In *Proceedings of NAACL.*

Botha, J., and Blunsom, P. 2014. Compositional Morphology for Word Representations and Language Modelling. In *Proceedings of ICML.*

Botha, J. 2014. Probabilistic Modelling of Morphologically Rich Languages. *DPhil Dissertation, Oxford University.*

Chen, S., and Goodman, J. 1998. An Empirical Study of Smoothing Techniques for Language Modeling. *Technical Report, Harvard University.*

Cheng, W. C.; Kok, S.; Pham, H. V.; Chieu, H. L.; and Chai, K. M. 2014. Language Modeling with Sum-Product Networks. In *Proceedings of INTERSPEECH.*

Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of EMNLP.*

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural Language Processing (almost) from Scratch. *Journal of Machine Learning Research* 12:2493–2537.

Creutz, M., and Lagus, K. 2007. Unsupervised Models for Morpheme Segmentation and Morphology Learning. In *Proceedings of the ACM Transations on Speech and Language Processing.*

Deerwester, S.; Dumais, S.; and Harshman, R. 1990. Indexing by Latent Semantic Analysis. *Journal of American Society of Information Science* 41:391–407.

dos Santos, C. N., and Guimaraes, V. 2015. Boosting Named Entity Recognition with Neural Character Embeddings. In *Proceedings of ACL Named Entities Workshop.*

dos Santos, C. N., and Zadrozny, B. 2014. Learning Character-level Representations for Part-of-Speech Tagging. In *Proceedings of ICML.*

Graves, A. 2013. Generating Sequences with Recurrent Neural Networks. *arXiv:1308.0850.*

Hinton, G.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2012. Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. *arxiv:1207.0580.*

Hochreiter, S., and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation* 9:1735–1780.

Kalchbrenner, N.; Grefenstette, E.; and Blunsom, P. 2014. A Convolutional Neural Network for Modelling Sentences. In *Proceedings of ACL.*

Kim, Y. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of EMNLP.*

Krizhevsky, A.; Sutskever, I.; and Hinton, G. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of NIPS.*

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. D. 1989. Handwritten Digit Recognition with a Backpropagation Network. In *Proceedings of NIPS.*

Lei, T.; Barzilay, R.; and Jaakola, T. 2015. Molding CNNs for Text: Non-linear, Non-consecutive Convolutions. In *Proceedings of EMNLP.*

Ling, W.; Lui, T.; Marujo, L.; Astudillo, R. F.; Amir, S.; Dyer, C.; Black, A. W.; and Trancoso, I. 2015. Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. In *Proceedings of EMNLP.*

Luong, M.-T.; Socher, R.; and Manning, C. 2013. Better Word Representations with Recursive Neural Networks for Morphology. In *Proceedings of CoNLL.*

Marcus, M.; Santorini, B.; and Marcinkiewicz, M. 1993. Building a Large Annotated Corpus of English: the Penn Treebank. *Computational Linguistics* 19:331–330.

Mikolov, T., and Zweig, G. 2012. Context Dependent Recurrent Neural Network Language Model. In *Proceedings of SLT.*

Mikolov, T.; Karafiat, M.; Burget, L.; Cernocky, J.; and Khudanpur, S. 2010. Recurrent Neural Network Based Language Model. In *Proceedings of INTERSPEECH.*

Mikolov, T.; Deoras, A.; Kombrink, S.; Burget, L.; and Cernocky, J. 2011. Empirical Evaluation and Combination of Advanced Language Modeling Techniques. In *Proceedings of INTERSPEECH.*

Mikolov, T.; Sutskever, I.; Deoras, A.; Le, H.-S.; Kombrink, S.; and Cernocky, J. 2012. Subword Language Modeling with Neural Networks. *preprint: www.fit.vutbr.cz/imikolov/rnnlm/char.pdf.*

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781.*

Mnih, A., and Hinton, G. 2007. Three New Graphical Models for Statistical Language Modelling. In *Proceedings of ICML.*

Morin, F., and Bengio, Y. 2005. Hierarchical Probabilistic Neural Network Language Model. In *Proceedings of AISTATS.*

Pascanu, R.; Culcehre, C.; Cho, K.; and Bengio, Y. 2013. How to Construct Deep Neural Networks. *arXiv:1312.6026.*

Qui, S.; Cui, Q.; Bian, J.; and Gao, B. 2014. Co-learning of Word Representations and Morpheme Representations. In *Proceedings of COLING.*

Shen, Y.; He, X.; Gao, J.; Deng, L.; and Mesnil, G. 2014. A Latent Semantic Model with Convolutional-pooling Structure for Information Retrieval. In *Proceedings of CIKM.*

Srivastava, R. K.; Greff, K.; and Schmidhuber, J. 2015. Training Very Deep Networks. *arXiv:1507.06228.*

Sundermeyer, M.; Schluter, R.; and Ney, H. 2012. LSTM Neural Networks for Language Modeling.

Sutskever, I.; Martens, J.; and Hinton, G. 2011. Generating Text with Recurrent Neural Networks.

Sutskever, I.; Vinyals, O.; and Le, Q. 2014. Sequence to Sequence Learning with Neural Networks.

Wang, M.; Lu, Z.; Li, H.; Jiang, W.; and Liu, Q. 2015. *gen*CNN: A Convolutional Architecture for Word Sequence Prediction. In *Proceedings of ACL.*

Werbos, P. 1990. Back-propagation Through Time: what it does and how to do it. In *Proceedings of IEEE.*

Zaremba, W.; Sutskever, I.; and Vinyals, O. 2014. Recurrent Neural Network Regularization. *arXiv:1409.2329.*

Zhang, S.; Jiang, H.; Xu, M.; Hou, J.; and Dai, L. 2015. The Fixed-Size Ordinally-Forgetting Encoding Method for Neural Network Language Models. In *Proceedings of ACL.*

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level Convolutional Networks for Text Classification. In *Proceedings of NIPS.*