Efficient Contextual Representation Learning With Continuous Outputs

Liunian Harold Li[†], **Patrick H. Chen**[*], **Cho-Jui Hsieh**[*], **Kai-Wei Chang**[*]

[†]Peking University
[*]University of California, Los Angeles
liliunian@pku.edu.cn, patrickchen@g.ucla.edu
{chohsieh, kwchang}@cs.ucla.edu

Abstract

Contextual representation models have achieved great success in improving various downstream natural language processing tasks. However, these language-model-based encoders are difficult to train due to their large parameter size and high computational complexity. By carefully examining the training procedure, we observe that the softmax layer, which predicts a distribution of the target word, often induces significant overhead, especially when the vocabulary size is large. Therefore, we revisit the design of the output layer and consider directly predicting the pre-trained embedding of the target word for a given context. When applied to ELMo, the proposed approach achieves a 4-fold speedup and eliminates 80% trainable parameters while achieving competitive performance on downstream tasks. Further analysis shows that the approach maintains the speed advantage under various settings, even when the sentence encoder is scaled up.

1 Introduction

In recent years, text representation learning approaches, such as ELMo (Peters et al., 2018a), GPT (Radford et al., 2018), BERT (Devlin et al., 2019), and GPT-2 (Radford et al., 2019), have been developed to represent generic contextual information in natural languages by training an encoder with a language model objective on a large unlabelled corpus. During the training process, the encoder is given part of the text and asked to predict the missing pieces. Prior studies show that the encoders trained in this way can capture generic contextual information of the input text and improve a variety of downstream tasks significantly.

However, training contextual representations is known to be a resource-hungry process. For example, ELMo is reported to take about 2 weeks to train on a one-billion-token corpus with a vocabulary of 800,000 words using three GPUs.[1] This slow training procedure hinders the development cycle, prevents fine-grained parameter tuning, and makes training contextual representations inaccessible to the broader community. Recent work also raises concerns about the environmental implications of training such large models (Strubell et al., 2019). In addition, the success of these models stems from a large amount of data they used. It is challenging, if not impossible, to train a contextual representation model on a larger corpus with tens or hundreds of billions of tokens.

In this work, we explore how to accelerate contextual representation learning. We identify the softmax layer as the primary cause of inefficiency. This component takes up a considerable portion of all trainable parameters (80% for ELMo) and consumes a huge amount of training time. However, it is often not needed in the final model as the goal of contextual representation learning is to build a generic encoder. Therefore, it is rather a waste to allocate extensive computational resources to the softmax layer.

Inspired by Kumar and Tsvetkov (2019), we consider learning contextual representation models with continuous outputs. In the training process, the contextual encoder is learned by minimizing the distance between its output and a pre-trained target word embedding. The constant time complexity and small memory footprint of the output layer perfectly serve our desire to decouple learning contexts and words and devote most computational resources to the contextual encoder. In addition, we combine the approach with open-vocabulary word embeddings such that the model can be trained without the need to pre-define a

[1] https://github.com/allenai/bilm-tf/issues/55.

Transactions of the Association for Computational Linguistics, vol. 7, pp. 611–624, 2019. https://doi.org/10.1162/tacl_a_00289
Action Editor: Luke Zettlemoyer. Submission batch: 1/2019; Revision batch: 6/2019; Published 9/2019.

closed word set as the vocabulary. We also provide an alternative interpretation of learning contextual encoders with continuous outputs that sheds light on how the pre-trained embedding could affect the performance of the model.

We conduct a comprehensive empirical study to analyze the proposed approach and several existing methods that are originally proposed to reduce the complexity of the output layer in language models, such as the adaptive softmax, and the sub-word methods. We incorporate these approaches with ELMo and conduct a comprehensive study to compare them in terms of training speed and performance on five downstream tasks. We demonstrate that the proposed approach effectively reduces the training time and trainable parameters while maintaining competitive performance compared with the baselines. Our approach also exhibits consistent computational advanxtage under different conditions (e.g., with different vocabulary sizes, with different sentence encoders, and with different number of GPUs).

Source code is available at `https://github.com/uclanlp/ELMO-C`.

2 Background and Related Work

Contextual representation We review contextual representation models from two aspects: how they are trained and how they are used in downstream tasks.

CoVe (McCann et al., 2017) uses the source language encoder from a machine translation model as a contextual representation model. Peters et al. (2018a) advocate for the use of larger unlabelled corpora and proposes ELMo, a forward and a backward LSTM-based (Hochreiter and Schmidhuber, 1997) language model, whereas GPT (Radford et al., 2018) and GPT-2 (Radford et al., 2019) build a language model with the Transformer (Vaswani et al., 2017). BERT (Devlin et al., 2019) introduces the masked language model and provides deep bidirectional representation.

There are two existing strategies for applying pre-trained contextual representations to downstream tasks: 1) *feature-based* and 2) *fine-tuning*. In the *feature-based* approach, fixed features are extracted from the contextual encoder (e.g., ELMo, CoVe) and inserted as an input into a task-specific model. In the *fine-tuning* approach, the contextual encoder is designed as a part of the network architecture for downstream tasks, and its parameters are fine-tuned with the downstream task. BERT is designed for the *fine-tuning* approach but it is also evaluated with the *feature-based* approach. GPT-2 is a scaled-up version of GPT and exhibits strong performance under zero-shot settings.

Speeding up language models training Considerable efforts have been devoted to accelerating the training process of language models. One line of research focuses on developing faster sequence encoder architectures such as CNN (Kim et al., 2016; Dauphin et al., 2017), QRNN (Bradbury et al., 2016), SRU (Lei et al., 2018), and the Transformer (Vaswani et al., 2017). These architectures have been extensively used for learning language representations (Radford et al., 2018; Devlin et al., 2019; Tang et al., 2018). Another line of work focuses on the large-vocabulary issue, as a large and ever-growing vocabulary results in an intractable softmax layer. Our work falls into the second line and we review existing solutions in detail.

Several studies for language modeling focus on directly reducing the complexity of the softmax layer. Following Kumar and Tsvetkov (2019), we group them into two categories: sampling-based approximations and structural approximations. Sampling-based approximations include the sampled softmax (Bengio et al., 2003) and NCE (Mnih and Teh, 2012). The sampled softmax approximates the normalization term of softmax by sampling a subset of negative targets, and NCE replaces the softmax with a binary classifier. On the other hand, structural approximations such as the hierarchical softmax (Morin and Bengio, 2005) and the adaptive softmax (Grave et al., 2016), form a structural hierarchy to avoid expensive normalization. The adaptive softmax, in particular, groups words in the vocabulary into either a short-list or clusters of rare words. For frequent words, a softmax over the short-list would suffice, which reduces computation and memory usage significantly. The adaptive softmax has been shown to achieve results close to those of the full softmax while maintaining high GPU efficiency (Merity et al., 2018).

Regarding contextual representation models, ELMo used the sampled softmax and GPT and BERT resorted to a subword method. Specifically, they used WordPiece (Wu et al., 2016) or BPE (Sennrich et al., 2016) to split the words into

subwords and the language models were trained to take subwords as input and also predict subwords. This method is efficient and scalable, as the subword vocabulary can be kept small. One potential drawback of these subword-level language models, however, is that they produce representations for fragments of words. Therefore, it takes extra effort to generate word-level representations (see the discussion in Section 4.2).

The high cost of the softmax layer has also been noted in the sentence representation learning literature. Following the success of Word2Vec (Mikolov et al., 2013), methods such as SkipThought (Kiros et al., 2015) have been developed to learn distributed sentence representations by predicting the context sentences of a given sentence, which involves sequentially decoding words of the target sentences. Jernite et al. (2017) and Logeswaran and Lee (2018) notice the inefficiency of the softmax layer during decoding and propose to use discriminative instead of generative objectives, eliminating the need for decoding. However, these approaches are not directly applicable to contextual representation learning.

3 Approach

A contextual representation model, at its core, is a language model pre-trained on a large unlabeled corpus. In the following, we review the objective of language models and the architectures of existing contextual representation models. We then introduce the proposed model.

Language model objective Given a set of text sequences as the training corpus, we can construct a collection of word-context pairs (w, c), and the goal of a language model is to predict the word w based on the context c. In a forward language model, the context c is defined as the previous words in the sequence, whereas for a backward language model, the context of a word is defined as the following words. For a masked language model, some words in the input sentence are masked (e.g., replaced by a [MASK] token) and the objective is to predict the masked words from the remainder. Different contextual representation models optimize different objectives. For example, ELMo trains a forward and backward language model and BERT trains a masked-language model.

Model architecture A typical neural language model consists of three parts: 1) an input layer, 2) a sequence encoder, and 3) a softmax layer. Given a word-context pair (w, c), the input layer uses a word embedding or a character-CNN model (Kim et al., 2016) to convert the input words in c into word vectors. Then the sequence encoder embeds the context into a context vector $\boldsymbol{c} \in \mathcal{R}^m$ using a multi-layer LSTM (Hochreiter and Schmidhuber, 1997), a Gated CNN (Dauphin et al., 2017), or a Transformer (Vaswani et al., 2017). The softmax layer then multiplies the context vector $\boldsymbol{c}$ with an *output word embedding*[2] $\boldsymbol{W} \in \mathcal{R}^{V \times m}$ and uses a softmax function to produce a conditional distribution $p(w|c)$ over the vocabulary of size V.

In a language model, the learning objective $l(w, c)$ for (w, c) is then expressed as:

$$\begin{aligned} l(w,c) &= -\log p(w|c) \\ &= -\log \mathrm{softmax}(\boldsymbol{c}\boldsymbol{W}^{\boldsymbol{T}}) \\ &= -\boldsymbol{c}\cdot\boldsymbol{w} + \log \sum\nolimits_{w'} \exp(\boldsymbol{c}\cdot\boldsymbol{w'}), \quad (1) \end{aligned}$$

where $\boldsymbol{w} \in \mathcal{R}^m$ is a row from $\boldsymbol{W}$ corresponding to the target word w and the second term sums over the vocabulary. After the model is trained, the contextual representations are generated from the latent states of the sequence encoder. For example, ELMo combines the hidden states of the LSTMs to generate contextualized word embedding for each word in a sentence. We refer the reader to Peters et al. (2018a) for details.

Note that the size of $\boldsymbol{W}$ and the computational complexity of the second term in Eq. (1) scale linearly to the vocabulary size, V. Therefore, when V is large, the softmax layer becomes the speed bottleneck.

Our approach The scaling issue of softmax also occurs in other language generation and sequence-to-sequence models. In the literature, several approaches have been proposed to approximate the softmax layer or bypass it with a subword method (see Section 2). Recently, Kumar and Tsvetkov (2019) propose to treat the context vector as continuous outputs and directly minimize the distance

[2]The dimension of the original output from the sequence encoder may not match the dimension of the output word embedding. In that case, a projection layer is added after the original sequence encoder to ensure that the two dimensions match.

between the context vector and the pre-trained word embedding associated with the target word,

$$l(w, c) = d(\boldsymbol{c}, \boldsymbol{w}) \tag{2}$$

The distance function l could be the L2 distance $\|\boldsymbol{c} - \boldsymbol{w}\|_2$, the cosine distance $\frac{\boldsymbol{c} \cdot \boldsymbol{w}}{\|\boldsymbol{c}\| \|\boldsymbol{w}\|}$ or a probabilistic distance metric.

We argue that the idea of learning with continuous outputs particularly suits contextual representation learning. As the goal is to obtain a strong contextual encoder, it makes sense to use a pre-trained output word embedding and decouple learning the contextual encoder and the output embedding. In the remainder of this section, we discuss the computational efficiency of the proposed approach and its combination with the open-vocabulary word embedding. We also provide an alternative way to interpret training contextual encoders with continuous outputs.

3.1 Computational Efficiency

The continuous output layer has a reduced arithmetic complexity and trainable parameter size. We illustrate these improvements and how they contribute to reducing the overall training time of a contextual representation model in the following. For comparison, we include the sampled softmax, the adaptive softmax, and the subword method in the discussion.

3.1.1 Learning with Continue Outputs

Arithmetic complexity The arithmetic complexity (i.e., FLOPs) of evaluating loss with continue outputs (i.e., Eq. 2) takes $O(m)$, as we only need to calculate the distance between two m-dimensional vectors. The complexity of the sampled softmax is proportional to the number of negative samples per batch. When the vocabulary is huge, a large number of negative samples are needed (Jozefowicz et al., 2016). For the adaptive softmax, the time complexity is determined by the capacities of the short-list and the rare-word clusters, which grows sub-linearly to the vocabulary size. The complexity of the subword method is determined by the subword vocabulary size. In contrast, the time spent on the continuous output layer and loss evaluation remains constant with respect to the vocabulary size and is negligible.

Trainable parameter size The output word embedding usually takes up a huge part of the parameters of a language model. For example, the softmax layer in ELMo trained on the One Billion Word Benchmark (Chelba et al., 2013) takes up more than 80% of the trainable parameters of the entire model. Even if an approximation such as the sampled softmax is used, the number of trainable parameters is not reduced. Approaches like the adaptive softmax reduce the dimension of softmax embedding for rare words, the trainable parameter size of which is effectively reduced but still remains sizable. For a model trained on the same corpus (Grave et al., 2016), the adaptive softmax still amounts to 240 million parameters whereas the sequence encoder has only around 50 million parameters. On the contrary, we learn a contextual encoder with Eq. (2) using a pre-trained word embedding, reducing the trainable parameters besides the encoder from tens or hundreds of millions to zero.

3.1.2 Overall Training Time

We now discuss how the efficiency improvements to the output layer contribute to the reduction of the overall training time, in the context of synchronous stochastic gradient descent training on multiple GPUs. In general, the following three factors determine the training time.

Arithmetic complexity The arithmetic complexity of a model includes the complexity of the forward and backward propagation on the input layer, the sequence encoder, and the output layer. It also includes the overhead of the optimization algorithm such as gradient clipping and model updates. The complexity of this optimization overhead is often proportional to the number of parameters that need updating. With the continuous output layer, not only the arithmetic complexity but also the optimization overhead are reduced.

GPU memory consumption The training time is also affected by GPU memory consumption, as less GPU memory consumption leads to larger batch size. For the same amount of data and hardware resource, larger batch size means better parallelism and less training time. Our approach exhibits small GPU memory footprints, due to reductions of the arithmetic complexity (with fewer intermediate results to keep) and trainable parameter size (with fewer parameters to store). As a result, training with continuous outputs is 2 to 4 times more memory-efficient than with the softmax layer (see Section 5.2).

Note that as the output word embedding is fixed, we can keep that embedding in the main memory and only load the required part to the GPU memory. Despite the fact that this comes with an overhead of moving part of the output word embedding from CPU to GPU memory at each iteration, the benefit of parallelism often dominates over the communication overhead on mainstream hardware, where the GPU memory is often comparatively limited. We also note that larger batch size may lead to difficulty in optimization. Several methods have been developed to ease the large-batch training issue (Goyal et al., 2017; You et al., 2018). We show that these methods are sufficient for resolving the optimization difficulty in our experiment (Section 4).

Communication cost To train large neural network models, using multiple GPUs almost becomes a necessity. In addition, one way to scale up current systems is to increase the number of GPUs used. In such cases, the communication cost across GPUs needs to be taken into consideration. The cost occurs from synchronizing the parameters and their gradients across GPUs, which is proportional to the size of parameters that need to be updated. For the sampled softmax, due to the use of the sparse gradient, the communication cost is proportional to the number of the sampled words. For the adaptive softmax and the subword language model, the communication cost is proportional to the trainable parameter size. The continuous output layer, on the other hand, incurs little communication cost across GPUs.

3.2 Open-Vocabulary Training

We utilize the open-vocabulary word embedding as both the input and output layer embedding. Open-vocabulary word embeddings, such as the FastText embedding and the MIMICK model (Pinter et al., 2017), utilize character or subword information to provide word embeddings. They could represent an unlimited number of words with a fixed number of parameters. As a result, we can train contextual encoders with an open vocabulary, which means we do not need to pre-define a closed word set as the vocabulary and the model can be trained on any sequences of words.

Open-vocabulary input layer To be easily applied in various tasks, the contextual encoder usually has an open-vocabulary input layer. ELMo uses a character-CNN but it is relatively slow. Thus we use a pre-trained open-vocabulary word embedding as the input layer of the contextual encoder, reducing the time complexity of the input layer to a negligible level. This also aligns with the main spirit of our approach, which is to spend computational resources on the most important part, the sequence encoder.

Open-vocabulary output layer For the softmax layer, including efficient variants such as the adaptive softmax, the output vocabulary has to be pre-defined so that the normalization term can be calculated. As the softmax layer's arithmetic complexity and parameter size grow when the vocabulary size grows, the vocabulary is often truncated to avoid expensive computation.

With the continuous output layer, we can conduct training on an arbitrary sequence of words, as long as the output embedding for those words can be derived. This can be achieved by using the open-vocabulary embedding. This feature is especially attractive if we are training on domains or languages with a long-tail word distribution such as the biomedical domain, where truncating the vocabulary may not be acceptable.

3.3 Interpretation of Learning Contextual Encoders with Continuous Outputs

In the following, we justify the intuition behind learning with continue outputs and discuss how the pre-trained word embedding affects the performance of the model.

Language models are essentially modeling the word-context conditional probability matrix, that is, $\boldsymbol{A} \in \mathcal{R}^{N \times V}$ where $A_{c,w} = p(w|c)$, N is the number of all possible contexts, and V is the vocabulary size (Levy and Goldberg, 2014; Yang et al., 2017). The continuous output layer can be viewed as modeling $\boldsymbol{A}$ after using the word embedding as a projection matrix.

To illustrate this, consider the global objective of the layer with the cosine distance:[3]

$$\begin{aligned}\mathcal{L} &= \sum\nolimits_{(w,c)} \#(w,c) l(w,c) \\ &= -\sum\nolimits_{(w,c)} \#(w,c) \boldsymbol{c} \cdot \boldsymbol{w} \\ &= -\sum\nolimits_{c} \#(c) \boldsymbol{c} \cdot \sum_{w} p(w|c) \boldsymbol{w}, \\ &= -\sum\nolimits_{c} \#(c) \boldsymbol{c} \cdot \sum_{w} \boldsymbol{A}_{c,w} \boldsymbol{w},\end{aligned}$$

[3]For simplicity, we take the cosine distance as a running example but the conclusions hold for other distance functions.

Model	Input	Sequence Encoder	Output
ELMo	CNN	LSTM	Sampled Softmax
ELMo-*C* (ours)	FastText$_{cc}$	LSTM w/ LN	Cont w/ FastText$_{cc}$
ELMo-*A*	FastText$_{cc}$	LSTM w/ LN	Adaptive Softmax
ELMo-*Sub*	Subword	LSTM w/ LN	Softmax
ELMo-C_{OneB}	FastText$_{\text{OneB}}$	LSTM w/ LN	Cont w/ FastText$_{\text{OneB}}$
ELMo-C_{Rnd}	FastText$_{cc}$	LSTM w/ LN	Cont w/ Random Embedding
ELMo-C_{CNN}	Trained CNN	LSTM w/ LN	Cont w/ Trained CNN
ELMo-$C_{\text{CNN-CC}}$	Trained CNN	LSTM w/ LN	Cont w/ FastText$_{cc}$
ELMo-$C_{\text{CC-CNN}}$	FastText$_{cc}$	LSTM w/ LN	Cont w/ Trained CNN

Table 1: Specifications of variants of ELMo models compared in Sections 4 and 5. Cont means the model has continuous outputs. LN means layer normalization.

where $\#(w, c)$ is the number of occurrences of the pair (w, c) in the corpus. We assume all vectors ($\boldsymbol{c}$ and $\boldsymbol{w}$) are normalized.

To optimize the inner product between $\boldsymbol{c}$ and $\sum_w p(w|c)\boldsymbol{w}$, we essentially align the direction of context vector $\boldsymbol{c}$ with the expected word vector under context c, $\sum_w p(w|c)\boldsymbol{w} = E_{w\sim p(w|c)}\boldsymbol{w}$. In other words, given a word embedding matrix $\boldsymbol{W} \in R^{V\times d}$, our approach aligns $\boldsymbol{c}$ with the corresponding row $(\boldsymbol{AW})_{c,:}$ in $\boldsymbol{AW}$. Therefore, the objective can be viewed as conducting multivariate regression to approximate $(\boldsymbol{AW})_{c,:}$ given the context.

Based on this view, the performance of the contextual representation model depends on how much information of the original matrix $\boldsymbol{A}$ is preserved after projection with $\boldsymbol{W}$. In the spirit of PCA (Jolliffe, 2011), to keep the variance of $\boldsymbol{A}$, we would like to have $(\boldsymbol{AW})_{c,:}$ and $(\boldsymbol{AW})_{c',:}$ distant from each other if c and c' are very different. Therefore, a pre-trained word embedding, which projects words with different meanings into different positions in space, is a natural choice for the projection matrix $\boldsymbol{W}$ and can help preserve much of the variance of $\boldsymbol{A}$.

4 Experiment

We accelerate ELMo with the proposed approach and show that the resulting model ELMo-*C* is computationally efficient and maintains competitive performance, compared to the original ELMo model (ELMo), an ELMo variant with the adaptive softmax (ELMo-*A*[4]), and another variant with the subword method (ELMo-*Sub*).

[4]We include ELMo-*A* instead of a model with sampled softmax because the adaptive softmax has been shown to have superior performance (Grave et al., 2016).

4.1 Setup

Models In the following, we introduce the models in detail. Table 1 provides a summary. The original ELMo consists of a character-CNN as the input layer, a forward and backward LSTM with projection as the sequence encoder, and a sampled softmax as the output layer. Adagrad (Duchi et al., 2011) is used as the optimizer. We conduct experiments using the reimplementation of ELMo in AllenNLP (Gardner et al., 2018) and build the others upon it.

The key difference between ELMo-*C* and ELMo is that ELMo-*C* produces continuous outputs and we train it with the cosine distance loss. A FastText embedding trained on Common Crawl (Mikolov et al., 2017) (FastText$_{cc}$) is used as the output embedding. Based on preliminary experiments, we also make three minor changes: 1) we use FastText$_{cc}$ as the input layer as it is more efficient than the character-CNN model; 2) we add a layer norm (Ba et al., 2016) after the projection layer of the LSTM to improve the convergence speed; 3) we use Adam with the learning rate schedule from Chen et al. (2018) to help training with a large batch size.

Our main goal is to study how different output layers affect the training speed and performance, which cannot be achieved by just comparing ELMo-*C* and ELMo, due to the aforementioned minor changes to ELMo-*C*. Therefore, we introduce two additional baseline models (ELMo-*A* and ELMo-*Sub*), which differ from ELMo-*C* in a minimal way. Specifically, their sequence encoders and training recipes are kept the same as ELMo-*C*. Thus ELMo-*C*, ELMo-*A*, and ELMo-*Sub* can be directly compared.

	ELMo$_{\text{ORG}}$	BASE	FASTTEXT$_{\text{CC}}$	ELMO	ELMO-*A*	ELMO-*Sub*	ELMO-*C*
Time	–	–	–	14 x 3	5.7 x 4	3.9 x 4	2.5 x 4
Batch	–	–	–	128	256	320	768
Params	–	–	–	499M	196M	92M	76M
SNLI	88.7	88.0	87.7	88.5	88.9	87.1	88.8
Coref	NA	NA	68.90	72.9	72.9	72.4	72.9
SST-5	54.7	51.4	51.30 ± 0.77	52.96 ± 2.26	53.58 ± 0.77	53.02 ± 2.08	53.80 ± 0.73
NER	92.22	90.15	90.97 ± 0.43	92.51 ± 0.28	92.28 ± 0.20	92.17 ± 0.56	92.24 ± 0.10
SRL	84.6	81.4	80.2	83.4	82.7	82.4	82.4

Table 2: Computational efficiency of the main competing models and their performance on five NLP benchmarks. Time is the overall training time in Days x Cards format. Batch is the maximal batch size per card. Params is the number of trainable parameters in millions. Due to the small test sizes for NER and SST-5, we report mean and standard deviation across three runs. Our approach (ELMO-*C*) exhibits better computational efficiency and shows comparable performance compared with ELMO, ELMO-*A*, and ELMO-*Sub*.

ELMO-*A* uses the adaptive softmax as its output layer. We carefully choose the hyper-parameters of the adaptive softmax to obtain an efficient yet strong baseline. It has only half of the parameters of the one reported in Grave et al. (2016) but achieves a perplexity of 35.8, lower than ELMO's 39.7.

ELMO-*Sub* takes subwords as input and also predicts subwords. Thus, unlike other models, its vocabulary consists of around 30,000 subwords created using BPE (Sennrich et al., 2016). For this reason, a lookup-table-style embedding rather than FASTTEXT$_{\text{CC}}$ is used as its input layer and a vanilla softmax is used as its output layer. Its input and output word embedding are tied and trained from scratch.

For reference, we also list the results of ELMO and the baseline reported in Peters et al. (2018a) as ELMO$_{\text{ORG}}$ and BASE. However, these models are evaluated using different configurations. Finally, we also include FASTTEXT$_{\text{CC}}$ a (non-contextual) word embedding model, as another baseline.

All contextual representation models are trained on the One Billion Word Benchmark for 10 epochs and the experiments are conducted on a workstation with 8 GeForce GTX 1080Ti GPUs, 40 Intel Xeon E5 CPUs, and 128G main memory.

Downstream benchmarks We follow Peters et al. (2018a) and use the *feature-based* approach to evaluate contextual representations on downstream benchmarks. ELMO was evaluated on six benchmarks and we conduct evaluations on five of them. SQuAD (Rajpurkar et al., 2016) is not available for implementation reasons.[5] In the following, we briefly review the benchmarks and task-specific models. For details please refer to Peters et al. (2018a).

- **SNLI** (Bowman et al., 2015): The textual entailment task seeks to determine whether a ''hypothesis'' can be entailed from a ''premise''. The task-specific model is ESIM (Chen et al., 2017).
- **Coref**: Coreference resolution is the task of clustering mentions in text that refer to the same underlying entities. The data set is from CoNLL 2012 shared task (Pradhan et al., 2012) and the model is from Lee et al. (2018). Note that we use an improved version of the Coref system (Lee et al., 2017) used in Peters et al. (2018a).
- **SST-5** (Socher et al., 2013): The task involves selecting one of five labels to describe a sentence from a movie review. We use the BCN model from McCann et al. (2017).
- **NER**: The CoNLL 2003 NER task (Sang and De Meulder, 2003) consists of newswire from the Reuters RCV1 corpus tagged with four different entity types. The model is a biLSTM-CRF from Peters et al. (2018a), similar to Collobert et al. (2011).
- **SRL**: Semantic role labeling models the predicate-argument structure of a sentence. It

[5]The SQuAD experiment in Peters et al. (2018a) was conducted with an implementation in TensorFlow. The experiment setting is not currently available in AllenNLP (`https://github.com/allenai/allennlp/pull/1626`), nor can it be easily replicated in PyTorch.

seeks to answer ''Who did what to whom''. The model is from He et al. (2017) and the data set is from Pradhan et al. (2013).

For SNLI, SST-5, NER, and SRL, we use the same downstream models as in Peters et al. (2018a) re-implemented in AllenNLP.[6] For Coref, Peters et al. (2018a) uses the model from Lee et al. (2017) and we use an improved model (Lee et al., 2018) from the same authors. For all the tasks, the hyper-parameters are tuned to maximize the performance for the original ELMo and all models are tested under the same configurations.

4.2 Main Results

We report the main results in Table 2. Our approach (ELMo-*C*) enjoys a substantial computational advantage while maintaining competitive or even superior performance, compared to ELMo, ELMo-*A*, and ELMo-*Sub*.

Model efficiency For model efficiency, the statistics of ELMo is reported by the original authors and they use three GTX 1080 Tis. We train ELMo-*A*, ELMo-*Sub*, and ELMo-*C* using four GTX 1080 Tis. Roughly speaking, compared with ELMo, ELMo-*C* is 4.2x faster and 6x more memory-efficient. To give a clear view of the speedup the CONT layer brings, we compare ELMo-*C* with ELMo-*A*. ELMo-*A* differs from ELMo-*C* only in the output layer. Still, ELMo-*C* has a 2.28x speed advantage and is 3x more memory-efficient. Compared with ELMo-*Sub*, our approach holds a 1.56x speed advantage and is 2x more memory-efficient. The results here only show the overall efficiency of our approach under the setting of ELMo and a detailed analysis of the efficiency is desirable, which we provide in Section 5.2.

Performance on downstream tasks ELMo-*C* works especially well on semantic-centric tasks, such as SNLI, Coref, and SST-5. However, for tasks that required a certain level of syntactic information, such as NER and SRL (He et al., 2018), ELMo-*C* suffers from slight performance degradation compared with ELMo, but it remains competitive with ELMo-*A* and ELMo-*Sub*. We suspect that the performance degradation is related to the pre-trained embedding and conduct further analysis in Section 5.1.

[6]For SRL, the score reported by AllenNLP is lower than the score reported by CoNLL official script.

In addition, we notice that the performance of ELMo-*Sub* is unstable. It shows satisfying performance on SST-5, NER, and SRL. However, it lags behind on Coref and even fails to outperform $\text{FASTTEXT}_{\text{cc}}$ on SNLI. ELMo-*Sub* provides subword-level contextual representations, which we suspect could be the cause of unstable performance. Specifically, to get the representation for a word in evaluation on word-level tasks, we follow Devlin et al. (2019) to use the representation of its first subword. This could be sub-optimal if precise word-level representation is desired (e.g., the suffix of a word is an important feature). These results are consistent with the observation in Kitaev and Klein (2018). They find that special design has to be made to apply BERT to constituency parsing because of the subword segmentation. However, we notice that the scope of our experiment is limited. It is likely that when ELMo-*Sub* is scaled up or used with the *fine-tuning* method, the aforementioned issue is alleviated—we leave that to future work.

5 Analysis

We conduct analysis regarding the effect of the pre-trained word embedding on the performance of the contextual encoder. We also investigate the contributions of different factors to the overall training time and study the speedup of our approach under various conditions.

5.1 Effect of the Pre-trained Embedding

We show the effect of the pre-trained embedding by introducing several variants of ELMo-*C* (summarized in Table 1) and list their performance in Table 3.

Quality of the pre-trained embedding We aim to understand how the quality of the pre-trained output word embedding $\boldsymbol{W}$ affects the performance of the contextual encoder. To study this, we train a FastText word embedding on the One Billion Word Benchmark, a much smaller corpus than Common Crawl. We then train an ELMo-*C* variant, ELMo-C_{ONEB}, by using this embedding in the input and output layers. Comparing it to ELMo-*C*, ELMo-C_{ONEB} holds up surprisingly well and it is competitive on SNLI, Coref, and SST-5 while being inferior on NER and SRL.

This motivates us to take a step further and use a completely random output word embedding.

Task	ELMo-C	ELMo-C_{ONEB}	ELMo-C_{RND}	ELMo-C_{CNN}	ELMo-$C_{\text{CNN-CC}}$	ELMo-$C_{\text{CC-CNN}}$
SNLI	88.8	88.4	88.4	88.2	88.0	88.4
Coref	72.9	73.0	72.4	72.9	72.8	72.6
SST-5	53.80 ± 0.73	52.70 ± 0.90	53.01 ± 1.67	53.38 ± 0.68	54.33 ± 1.26	54.16 ± 0.96
NER	92.24 ± 0.10	92.03 ± 0.47	91.99 ± 0.35	92.24 ± 0.36	92.04 ± 0.33	91.93 ± 0.53
SRL	82.4	82.2	82.9	82.8	83.4	83.3

Table 3: Performance of ablation models on five NLP benchmarks. ELMo-C is included for reference.

We replace the output embedding of ELMo-C with a random embedding matrix, of which each element is randomly drawn from a standard normal distribution. We denote this model as ELMo-C_{RND}. We find that this model performs well (Table 3), with only a mild performance drop compared to ELMo-C. The performance of ELMo-C_{RND} shows the robustness of the proposed approach and demonstrates that the deep LSTM is expressive enough to fit a complex output space. However, we find that the pre-trained input word embedding is still indispensable because using a randomly initialized input embedding would lead to brittle performance (e.g., 85.8 on SNLI).

Pre-trained CNN layer as word embedding In Section 4, we observed that models using FastText embedding (ELMo-C and ELMo-A) as input performed worse than ELMo on SRL, a task relying heavily on syntactic information. We suspect that the FastText embedding is weaker on capturing syntactic information than the character-CNN trained in ELMo (Peters et al., 2018b). To verify this, we train ELMo-C using the trained CNN layer from ELMo as the input layer (ELMo-$C_{\text{CNN-CC}}$) or the output embedding (ELMo-$C_{\text{CC-CNN}}$). We observe that the two models exhibit notably better performance on SRL (see Table 3). We also consider a ELMo-C_{CNN} model, which uses the CNN layer as both the input and output embedding. On SRL, ELMo-C_{CNN} performs favorably compared to ELMo-C but slightly worse than ELMo-$C_{\text{CNN-CC}}$ or ELMo-$C_{\text{CC-CNN}}$. We suspect that this is because ELMo-$C_{\text{CNN-CC}}$ and ELMo-$C_{\text{CC-CNN}}$ benefit from different kinds of embeddings in the input layer and the output layer.

5.2 Computational Efficiency

Next, we study the computational efficiency of the continuous output layer against several baselines from two aspects. First, in Section 3.1, we discussed three factors governing the overall training time of the model: 1) arithmetic complexity, 2) GPU memory consumption, and 3) communication cost. We aim to study *how each factor affects the overall training time of each model.* Second, in the above experiments, we focus on ELMo with the LSTM as the sequence encoder. We wonder *whether the continuous output layer can deliver attractive speedup for sequence encoders of different types and sizes*.

We investigate the continuous output layer (CONT) and three common baseline output layers: 1) the subword-level language model (SUBWORD), 2) the adaptive softmax layer (ADAPTIVE), and 3) the sampled softmax layer (SAMPLED). Additionally, we include a variant of the sampled softmax denoted as FIXED where the output word embedding is initialized by the FastText embedding and fixed during the training. This output layer is similar to a special case of CONT with a ranking loss, where the model encourages its output to be close to the target word embedding but far from a negative sample.

In total, we study five different output layers. For several output layers, the trade-off between computational efficiency and model performance is controlled by their hyper-parameters. We choose hyper-parameters close to those reported in the literature to strike a balance between speed and performance.

5.2.1 Speedup Breakdown

We pair the five different output layers with the same input layer (fixed word embedding) and sequence encoder (ELMo's LSTM with projection). We then test the training speed of these models under three scenarios, which are designed to reflect the individual effect of the arithmetic complexity, the GPU memory consumption, and the communication cost:

- **S1 (small batch)**: We use one GPU card and set the batch size to be 1. The asynchronous execution feature of the GPU is disabled. The time needed to finish one batch is reported.

	Vocab	Params	Batch	S1 (small batch)	S2 (large batch)	S3 (multiple GPUs)
CONT	∞	76M	640	0.47s	115.28s	34.58s
FIXED	∞	76M	512	1.17x	1.24x	1.24x
SUBWORD	∞	92M	320	1.09x	1.53x	1.55x
ADAPTIVE	40K	97M	384	1.08x	1.30x	1.34x
	800K	196M	256	1.16x	1.47x	1.89x
	2000K	213M	192	1.25x	1.82x	2.49x
SAMPLED	40K	96M	512	1.07x	1.18x	1.30x
	800K	483M	256	1.15x	1.35x	1.91x
	2000K	1102M	64	1.16x	2.35x	16.09x

Table 4: Statistics on the computation efficiency of different models. For CONT, we report the actual training time in seconds. For other models, we report the relative training time compared to CONT. Params: Number of trainable parameters of the whole model in millions. Batch: Maximal batch size per card.

- **S2 (large batch)**: We use one GPU card and the maximal batch size. The time needed to finish training on one million words for each model is reported.
- **S3 (multiple GPUs)**: We use 4 GPU cards and the maximal batch size. The time needed to finish training on one million words for each model is reported.

In Table 4, we report the training speed of the models under each scenario.[7] In addition, we report the parameter size and the maximal batch size on one GPU card. For ADAPTIVE and SAMPLED, the vocabulary size also affects the training speed so we test them under three different vocabulary sizes:[8] 40K, 800K, and 2,000K.

Arithmetic complexity The arithmetic complexity of the models is reflected by the speed under S1, where the GPU memory is always abundant and the arithmetic complexity is the dominating factor. CONT holds a mild advantage (1.07x-1.25x) over baseline models, which is expected because the LSTM layers in ELMo are quite slow and that undermines the advantage of the continuous output layer. For ELMO-*Sub*, the small yet non-negligible softmax layer adds overhead to the arithmetic complexity. FIXED, ADAPTIVE, and SAMPLED have similar arithmetic complexity but ADAPTIVE has the highest complexity when the vocabulary size is large (e.g., 2,000K).

GPU memory consumption The effect of GPU memory consumption can be observed by comparing the statistics under S1 and S2. The difference between S2 and S1 is that the parallel computing of the GPU is fully utilized. For CONT, its great GPU memory efficiency helps it gain larger speedup under S2, especially against common baselines such as SUBWORD, ADAPTIVE, and SAMPLED. For ELMO-*Sub*, in addition to the overhead from the softmax layer, breaking words into subwords leads to longer sequences, which increases the training time by 1.1x. Thus it is 1.53x slower than CONT under S2. SAMPLED suffers from its huge parameter size and exhibits poor scalability with respect to the vocabulary size (2.35x slower when the vocabulary size reaches 2,000K).

Communication cost The effect of the communication cost across GPUs can be observed by comparing the statistics under S2 and S3. As the communication cost and GPU memory consumption both are highly dependent on the parameter size, the observations are similar.

[7] CONT under S3 is slightly slower than the ELMO-*C* model reported in Section 4.2. This is because when training the ELMO-*C* model reported in 4.2, we actually train a forward ELMO-*C* on two cards and train a backward ELMO-*C* on two other cards, which reduces the communication cost by half. This optimization is only applicable to our approach in the setting of ELMo and does not work for other baseline methods. In this experiment, we disable this optimization for generosity.

[8] The 2,000K vocabulary is created on the tokenized 250-billion-word Common Crawl corpus (Panchenko et al., 2017), which covers words that appear more than 397 times.

	LSTM	LSTMx2	Trans Base	ELMo	Trans Large	GPT
Cont	3.97s	10.42s	15.87s	34.58s	48.55s	43.53s
Fixed	1.93x	1.32x	1.52x	1.24x	1.37x	1.14x
Subword	2.32x	1.49x	1.78x	1.55x	1.72x	1.44x
Adaptive	4.58x	2.20x	2.62x	1.89x	3.28x	2.33x
Sampled	2.50x	1.60x	2.91x	1.91x	OOM	8.31x

Table 5: Time needed to finish training on one million words for each model using 4 GPU cards and the maximal batch size. For Cont, we report the actual training time in seconds. For other models, we report the relative training time compared to Cont. OOM means that the GPU memory is not sufficient. Cont shows substantial speedup over common baselines under all scenarios.

5.2.2 The Continuous Output Layer with Different Sequence Encoders

For this experiment, we pair the output layers with different sequence encoders and investigate their training time. We start from a single-layer LSTM with a hidden size of 2048 (LSTM) and a two-layer version (LSTMx2), both reported in Grave et al. (2016). They are all smaller than the sequence encoder used in ELMo. We then scale up to the forward and backward Transformer reported in Peters et al. (2018b) (Trans Base) and the multi-layer LSTM with projection in ELMo(ELMo). Finally, we test two larger Transformer, Trans Large, a scaled-up version of Trans Base, and a uni-directional Transformer (denoted as GPT) with the same size as $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019) and GPT (Radford et al., 2018), respectively. For all models but GPT, the lengths of the input sequences are fixed at 20. For GPT, we use input sequences of length 512, following its original setting. For Adaptive and Sampled, we fix the vocabulary size at 800K.

We report the training time of each model using four GPU cards and the maximal batch size (S3) in Table 5. We find that the continuous output layer remains attractive, even when the sequence encoder is as large as GPT. In that case, the speedup of Cont over Subword, Adaptive, and Sampled is still substantial (1.44x - 8.31x). In addition, we observe that for sequence encoders of the same type, more complex they get, less speedup Cont enjoys, which is expected. For instance, from LSTM to LSTMx2, the speedup of Cont decreases noticeably. However, the speedup the continuous output brings also depends on the architecture of the sequence encoder. For instance, though Trans Base and Trans Large are more complex than LSTMx2, Cont enjoys larger speedup with those transformers. Profiling the training process of sequence decoders such as LSTM and the Transformer on GPU devices is an interesting research topic but out of the scope of this study.

6 Conclusion

We introduced an efficient framework to learn contextual representation without the softmax layer. The experiments with ELMo showed that we significantly accelerate the training of the current models while maintaining competitive performance on various downstream tasks.

Acknowledgments

We wish to thank the anonymous reviewers, the editor, Mark Yatskar, Muhao Chen, Xianda Zhou, and members at UCLANLP lab for helpful comments. We also thank Yulia Tsvetkov and Sachin Kumar for help with implementing the continuous output layer as well as Jieyu Zhao, Kenton Lee, and Nelson Liu for providing reproducible source code for experiments. This work was supported by National Science Foundation grant IIS-1760523 and IIS-1901527.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Yoshua Bengio and Jean-Sébastien Senécal. 2003. Quick training of probabilistic neural nets by importance sampling. In *AISTATS*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2016. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *ACL*.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017. Enhanced LSTM for natural language inference. In *ACL*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *ICML*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-NLT*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: training Imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2016. Efficient softmax approximation for GPUs. *arXiv preprint arXiv:1609.04309*.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep semantic role labeling: What works and what's next. In *ACL*.

Shexia He, Zuchao Li, Hai Zhao, and Hongxiao Bai. 2018. Syntax for semantic role labeling, to be, or not to be. In *ACL*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:1735–1780.

Yacine Jernite, Samuel R Bowman, and David Sontag. 2017. Discourse-based objectives for fast unsupervised sentence representation learning. *arXiv preprint arXiv:1705.00557*.

Ian Jolliffe. 2011. Principal component analysis. In *International Encyclopedia of Statistical Science*, Springer Berlin Heidelberg.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *AAAI*.

Ryan Kiros, Yukun Zhu, Ruslan R. Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *NIPS*.

Nikita Kitaev and Dan Klein. 2018. Multilingual constituency parsing with self-attention and pre-training. *arXiv preprint arXiv:1812.11760*.

Sachin Kumar and Yulia Tsvetkov. 2019. Von Mises-Fisher loss for training sequence to sequence models with continuous outputs. In *ICLR*.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In *EMNLP*.

Kenton Lee, Luheng He, and Luke Zettlemoyer. 2018. Higher-order coreference resolution with coarse-to-fine inference. In *NAACL-HLT*.

Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. 2018. Simple recurrent units for highly parallelizable recurrence. In *EMNLP*.

Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*.

Lajanugen Logeswaran and Honglak Lee. 2018. An efficient framework for learning sentence representations. *ICLR*.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *NIPS*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. An analysis of neural language modeling at multiple scales. *arXiv preprint arXiv:1803.08240*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2017. Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*.

A Mnih and YW Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. In *ICML*.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *AISTATS*.

Alexander Panchenko, Eugen Ruppert, Stefano Faralli, Simone Paolo Ponzetto, and Chris Biemann. 2017. Building a web-scale dependency-parsed corpus from CommonCrawl. *arXiv preprint arXiv:1710.01779*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018a. Deep contextualized word representations. In *NAACL-HLT*.

Matthew E. Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. 2018b. Dissecting contextual word embeddings: Architecture and representation. In *EMNLP*.

Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. 2017. Mimicking word embeddings using subword RNNs. In *EMNLP*.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Björkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong. 2013. Towards robust linguistic analysis using OntoNotes. In *CoNLL*.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint Conference on EMNLP and CoNLL-Shared Task*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *OpenAI Blog*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100, 000+ questions for machine comprehension of text. In *EMNLP*.

Erik F Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *ACL*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*.

Shuai Tang, Hailin Jin, Chen Fang, Zhaowen Wang, and Virginia de Sa. 2018. Speeding up context-based sentence representation learning with non-autoregressive convolutional decoding. In *Workshop on Representation Learning for NLP*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and James A. Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. 2017. Breaking the softmax bottleneck: A high-rank RNN language model. *arXiv preprint arXiv:1711.03953*.

Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. 2018. ImageNet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing, ICPP 2018*.