

Natural Language Processing with Small Feed-Forward Networks

Jan A. Botha Emily Pitler Ji Ma Anton Bakalov

Alex Salcianu David Weiss Ryan McDonald Slav Petrov

Google Inc.

Mountain View, CA

{jabot,epitler,maji,abakalov,salcianu,djweiss,ryanmcd,slav}@google.com

Abstract

We show that small and shallow feed-forward neural networks can achieve near state-of-the-art results on a range of unstructured and structured language processing tasks while being considerably cheaper in memory and computational requirements than deep recurrent models. Motivated by resource-constrained environments like mobile phones, we showcase simple techniques for obtaining such small neural network models, and investigate different tradeoffs when deciding how to allocate a small memory budget.

1 Introduction

Deep and recurrent neural networks with large network capacity have become increasingly accurate for challenging language processing tasks. For example, machine translation models have been able to attain impressive accuracies, with models that use hundreds of millions (Bahdanau et al., 2014; Wu et al., 2016) or billions (Shazeer et al., 2017) of parameters. These models, however, may not be feasible in all computational settings. In particular, models running on mobile devices are often constrained in terms of memory and computation.

Long Short-Term Memory (LSTM) models (Hochreiter and Schmidhuber, 1997) have achieved good results with small memory footprints by using character-based input representations: e.g., the part-of-speech tagging models of Gillick et al. (2016) have only roughly 900,000 parameters. Latency, however, can still be an issue with LSTMs, due to the large number of matrix multiplications they require (eight per LSTM cell): Kim and Rush (2016) report speeds of only 8.8 words/second when running a two-layer LSTM translation system on an Android phone.

Feed-forward neural networks have the potential to be much faster. In this paper, we show that small feed-forward networks can achieve results at or near the state-of-the-art on a variety of natural language processing tasks, with an order of magnitude speedup over an LSTM-based approach.

We begin by introducing the network model structure and the character-based representations we use throughout all tasks (§2). The four tasks that we address are: language identification (Lang-ID), part-of-speech (POS) tagging, word segmentation, and reordering for translation. In order to use feed-forward networks for structured prediction tasks, we use transition systems (Titov and Henderson, 2007, 2010) with feature embeddings as proposed by Chen and Manning (2014), and introduce two novel transition systems for the last two tasks. We focus on *budgeted* models and ablate four techniques (one on each task) for improving accuracy for a given memory budget:

1. Quantization: Using more dimensions and less precision (Lang-ID: §3.1).
2. Word clusters: Reducing the network size to allow for word clusters and derived features (POS tagging: §3.2).
3. Selected features: Adding explicit feature conjunctions (segmentation: §3.3).
4. Pipelines: Introducing another task in a pipeline and allocating parameters to the auxiliary task instead (reordering: §3.4).

We achieve results at or near state-of-the-art with small (< 3 MB) models on all four tasks.

2 Small Feed-Forward Network Models

The network architectures are designed to limit the memory and runtime of the model. Figure 1 illustrates the model architecture:

1. Discrete features are organized into groups (e.g., $\mathbf{E}_{\text{bigrams}}$), with one embedding matrix $\mathbf{E}_g \in \mathbb{R}^{V_g \times D_g}$ per group.
2. Embeddings of features extracted for each group are reshaped into a single vector and concatenated to define the output of the embedding layer as $\mathbf{h}_0 = [\mathbf{X}_g \mathbf{E}_g \mid \forall g]$.
3. A single hidden layer, \mathbf{h}_1 , with M rectified linear units (Nair and Hinton, 2010) is fully connected to \mathbf{h}_0 .
4. A softmax function models the probability of an output class y : $P(y) \propto \exp(\beta_y^T \mathbf{h}_1 + b_y)$, where $\beta_y \in \mathbb{R}^M$ and b_y are the weight vector and bias, respectively.

Memory needs are dominated by the embedding matrix sizes ($\sum_g V_g D_g$, where V_g and D_g are the vocabulary sizes and dimensions respectively for each feature group g), while runtime is strongly influenced by the hidden layer dimensions.

Hashed Character n -grams Previous applications of this network structure used (pretrained) word embeddings to represent words (Chen and Manning, 2014; Weiss et al., 2015). However, for word embeddings to be effective, they usually need to cover large vocabularies (100,000+) and dimensions (50+). Inspired by the success of character-based representations (Ling et al., 2015), we use features defined over character n -grams instead of relying on word embeddings, and learn their embeddings from scratch.

We use a distinct feature group g for each n -gram length N , and control the size V_g directly by applying *random feature mixing* (Ganchev and Dredze, 2008). That is, we define the feature value v for an n -gram string x as $v = \mathcal{H}(x) \bmod V_g$, where \mathcal{H} is a well-behaved hash function. Typical values for V_g are in the 100-5000 range, which is far smaller than the exponential number of unique raw n -grams. A consequence of these small feature vocabularies is that we can also use small feature embeddings, typically $D_g=16$.

Quantization A commonly used strategy for compressing neural networks is quantization, using less precision to store parameters (Han et al., 2015). We compress the embedding weights (the vast majority of the parameters for these shallow models) by storing scale factors for each embedding (details in the supplementary material). In §3.1, we contrast devoting model size to higher

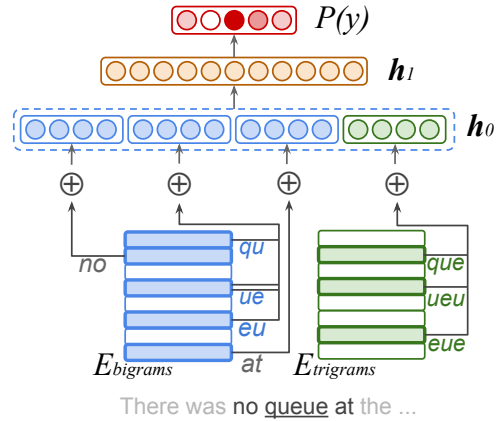


Figure 1: An example network structure for a model using bigrams of the previous, current and next word, and trigrams of the current word. Does not illustrate hashing.

precision and lower dimensionality versus lower precision and more network dimensions.

Training Our objective function combines the cross-entropy loss for model predictions relative to the ground truth with L2 regularization of the biases and hidden layer weights. For optimization, we use mini-batched averaged stochastic gradient descent with momentum (Bottou, 2010; Hinton, 2012) and exponentially decaying learning rates. The mini-batch size is fixed to 32 and we perform a grid search for the other hyperparameters, tuning against the task-specific evaluation metric on held-out data, with early stopping. Full feature templates and optimal hyperparameter settings are given in the supplementary material.

3 Experiments

We experiment with small feed-forward networks for four diverse NLP tasks: language identification, part-of-speech tagging, word segmentation, and preordering for statistical machine translation.

Evaluation Metrics In addition to standard task-specific quality metrics, our evaluations also consider model size and computational cost. We skirt implementation details by calculating size as the number of kilobytes (1KB=1024 bytes) needed to represent all model parameters and resources. We approximate the computational cost as the number of floating-point operations (FLOPs) performed for one forward pass through the network given an embedding vector \mathbf{h}_0 . This cost is dominated by the matrix multiplications to compute (unscaled) activation unit values, hence our metric excludes the non-linearities and softmax normal-

ization, but still accounts for the final layer logits. To ground this metric, we also provide indicative absolute speeds for each task, as measured on a modern workstation CPU (3.50GHz Intel Xeon E5-1650 v3).

3.1 Language Identification

Recent shared tasks on code-switching (Molina et al., 2016) and dialects (Malmasi et al., 2016) have generated renewed interest in language identification. We restrict our focus to single language identification across diverse languages, and compare to the work of Baldwin and Lui (2010) on predicting the language of Wikipedia text in 66 languages. For this task, we obtain the input \mathbf{h}_0 by separately *averaging* the embeddings for each n -gram length ($N = [1, 4]$), as summation did not produce good results.

Table 1 shows that we outperform the low-memory nearest-prototype model of Baldwin and Lui (2010). Their nearest neighbor model is the most accurate but its memory scales linearly with the size of the training data.

Moreover, we can apply quantization to the embedding matrix without hurting prediction accuracy: it is better to use less precision for each dimension, but to use more dimensions. Our subsequent models all use quantization. There is no noticeable variation in processing speed when performing dequantization on-the-fly at inference time. Our 16-dim Lang-ID model runs at 4450 documents/second (5.6 MB of text per second) on the preprocessed Wikipedia dataset.

Relationship to Compact Language Detector

These techniques back the open-source Compact Language Detector v3 (CLD3)¹ that runs in Google Chrome browsers.² Our experimental Lang-ID model uses the same overall architecture as CLD3, but uses a simpler feature set, less involved preprocessing, and covers fewer languages.

3.2 POS Tagging

We apply our model as an unstructured classifier to predict a POS tag for each token independently, and compare its performance to that of the byte-to-span (BTS) model (Gillick et al., 2016). BTS is a 4-layer LSTM network that maps a sequence of bytes to a sequence of labeled spans, such as tokens and their POS tags. Both approaches limit

Model	Micro F1	Size
Baldwin and Lui (2010): NN	90.2	-
Baldwin and Lui (2010): NP	87.0	-
Small FF, 6 dim	87.3	334 KB
Small FF, 16 dim	88.0	800 KB
Small FF, 16 dim, <i>quantized</i>	88.0	302 KB

Table 1: Language Identification. Quantization allows trading numerical precision for larger embeddings. The two models from Baldwin and Lui (2010) are the nearest neighbor (NN) and nearest prototype (NP) approaches.

Model	Acc.	Wts.	MB	Ops.
Gillick et al. (2016)	95.06	900k	-	6.63m
Small FF	94.76	241k	0.6	0.27m
+Clusters	95.56	261k	1.0	0.31m
$\frac{1}{2}$ Dim.	95.39	143k	0.7	0.18m

Table 2: POS tagging. Embedded word clusters improves accuracy and allows the use of smaller embedding dimensions.

model size by using small input vocabularies: byte values in the case of BTS, and hashed character n -grams and (optionally) cluster ids in our case.

Bloom Mapped Word Clusters It is well known that word clusters can be powerful features in linear models for a variety of tasks (Koo et al., 2008; Turian et al., 2010). Here, we show that they can also be useful in neural network models. However, naively introducing word cluster features drastically increases the amount of memory required, as a word-to-cluster mapping file with hundreds of thousands of entries can be several megabytes on its own.³ By representing word clusters with a Bloom map (Talbot and Talbot, 2008), a key-value based generalization of Bloom filters, we can reduce the space required by a factor of ~ 15 and use 300KB to (approximately) represent the clusters for 250,000 word types.

In order to compare against the monolingual setting of Gillick et al. (2016), we train models for the same set of 13 languages from the Universal Dependency treebanks v1.1 (Nivre et al., 2016) corpus, using the standard predefined splits.

As shown in Table 2, our best models are 0.3% more accurate on average across all languages than the BTS monolingual models, while using 6x fewer parameters and 36x fewer FLOPs. The cluster features play an important role, providing a 15% relative reduction in error over our vanilla model, but also increase the overall size. Halv-

¹github.com/google/cld3

²As of the date of this writing in 2017.

³For example, the commonly used English clusters from the BLLIP corpus is over 7 MB – people.csail.mit.edu/maestro/papers/bllip-clusters.gz

Transition	
SPLIT	$([\sigma], [i \beta]) \rightarrow ([\sigma i], [\beta])$
MERGE	$([\sigma], [i \beta]) \rightarrow ([\sigma], [\beta])$

Table 3: Segmentation Transition system. Initially all characters are on the buffer β and the stack σ is empty: $([], [c_1c_2\dots c_n])$. In the final state the buffer is empty and the stack contains the first character for each word.

ing all feature embedding dimensions (except for the cluster features) still gives a 12% reduction in error and trims the overall size back to 1.1x the vanilla model, staying well under 1MB in total. This halved model configuration has a throughput of 46k tokens/second, on average.

Two potential advantages of BTS are that it does not require tokenized input and has a more accurate multilingual version, achieving 95.85% accuracy. From a *memory* perspective, one multilingual BTS model will take less space than separate FF models. However, from a *runtime* perspective, a pipeline of our models doing language identification, word segmentation, and then POS tagging would still be faster than a single instance of the deep LSTM BTS model, by about 12x in our FLOPs estimate.⁴

3.3 Segmentation

Word segmentation is critical for processing Asian languages where words are not explicitly separated by spaces. Recently, neural networks have significantly improved segmentation accuracy (Zhang et al., 2016; Cai and Zhao, 2016; Liu et al., 2016; Yang et al., 2017; Kong et al., 2015). We use a structured model based on the transition system in Table 3, and similar to the one proposed by Zhang and Clark (2007). We conduct the segmentation experiments on the Chinese Treebank 6.0 with the recommended data splits. No external resources or pretrained embeddings are used. Hashing was detrimental to quality in our preliminary experiments, hence we do not use it for this task. To learn an embedding for unknown characters, we cast characters occurring only once in the training set to a special symbol.

Selected Features Because we are not using hashing here, we need to be careful about the size of the input vocabulary. The neural network with its non-linearity is in theory able to learn bigrams by conjoining unigrams, but it has been

⁴Our calculation of BTS FLOPs is very conservative and favorable to BTS, as detailed in the supplementary material.

Model	Accuracy	Size
Zhang et al. (2016)	95.01	—
Zhang et al. (2016)-combo	95.95	—
Small FF, 64 dim	94.24	846KB
Small FF, 256 dim	94.16	3.2MB
Small FF, 64 dim, <i>bigrams</i>	95.18	2.0MB

Table 4: Segmentation results. Explicit bigrams are useful.

Transition	Precondition
APPEND	$([\sigma i j], [\beta]) \rightarrow ([\sigma [i j]], [\beta])$
SHIFT	$([\sigma], [i \beta]) \rightarrow ([\sigma i], [\beta])$
SWAP	$([\sigma i j], [\beta]) \rightarrow [\sigma j], [i \beta]; i < j$

Table 5: Preordering Transition system. Initially all words are part of singleton spans on the buffer: $([], [[w_1][w_2]\dots[w_n]])$. In the final state the buffer is empty and the stack contains a single span.

shown that explicitly using character bigram features leads to better accuracy (Zhang et al., 2016; Pei et al., 2014). Zhang et al. (2016) suggests that embedding manually specified feature conjunctions further improves accuracy (‘Zhang et al. (2016)-combo’ in Table 4). However, such embeddings could easily lead to a model size explosion and thus are not considered in this work.

The results in Table 4 show that spending our memory budget on small bigram embeddings is more effective than on larger character embeddings, in terms of both accuracy and model size. Our model featuring bigrams runs at 110KB of text per second, or 39k tokens/second.

3.4 Preordering

Preordering source-side words into the target-side word order is a useful preprocessing task for statistical machine translation (Xia and McCord, 2004; Collins et al., 2005; Nakagawa, 2015; de Gispert et al., 2015). We propose a novel transition system for this task (Table 5), so that we can repeatedly apply a small network to produce these permutations. Inspired by a non-projective parsing transition system (Nivre, 2009), the system uses a SWAP action to permute spans. The system is sound for permutations: any derivation will end with all of the input words in a permuted order, and complete: all permutations are reachable (use SHIFT and SWAP operations to perform a bubble sort, then APPEND $n - 1$ times to form a single span). For training and evaluation, we use the English-Japanese manual word alignments from Nakagawa (2015).

Model	FRS	Size
Nakagawa (2015)	81.6	-
Small FF	75.2	0.5MB
Small FF + POS tags	81.3	1.3MB
Small FF + Tagger input fts.	76.6	3.7MB

Table 6: Preordering results for English \rightarrow Japanese. *FRS* (in $[0, 100]$) is the fuzzy reordering score (Talbot et al., 2011).

Pipelines For preordering, we experiment with either spending all of our memory budget on reordering, or spending some of the memory budget on features over predicted POS tags, which also requires an additional neural network to predict these tags. Full feature templates are in the supplementary material. As the POS tagger network uses features based on a three word window around the token, another possibility is to add all of the features that would have affected the POS tag of a token to the reorderer directly.

Table 6 shows results with or without using the predicted POS tags in the preorderer, as well as including the features used by the tagger in the reorderer directly and only training the downstream task. The preorderer that includes a separate network for POS tagging and then extracts features over the predicted tags is more accurate and smaller than the model that includes all the features that contribute to a POS tag in the reorderer directly. This pipeline processes 7k tokens/second when taking pretokenized text as input, with the POS tagger accounting for 23% of the computation time.

4 Conclusions

This paper shows that small feed-forward networks are sufficient to achieve useful accuracies on a variety of tasks. In resource-constrained environments, speed and memory are important metrics to optimize as well as accuracies. While large and deep recurrent models are likely to be the most accurate whenever they can be afforded, feed-forward networks can provide better value in terms of runtime and memory, and should be considered a strong baseline.

Acknowledgments

We thank Kuzman Ganchev, Fernando Pereira, and the anonymous reviewers for their useful comments.

References

- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. [Neural machine translation by jointly learning to align and translate](#). *arXiv preprint arXiv:1409.0473*.
- Timothy Baldwin and Marco Lui. 2010. [Language identification: The long and the short of the matter](#). In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 229–237. Association for Computational Linguistics.
- Léon Bottou. 2010. [Large-scale machine learning with stochastic gradient descent](#). In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.
- Deng Cai and Hai Zhao. 2016. [Neural word segmentation learning for Chinese](#). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 409–420, Berlin, Germany. Association for Computational Linguistics.
- Danqi Chen and Christopher D. Manning. 2014. [A fast and accurate dependency parser using neural networks](#). In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 740–750.
- M. Collins, P. Koehn, and I. Kučerová. 2005. [Clause restructuring for statistical machine translation](#). In *Proceedings of ACL*, pages 531–540.
- Kuzman Ganchev and Mark Dredze. 2008. [Small statistical models by random feature mixing](#). In *Proceedings of the ACL-08- HLT Workshop on Mobile Language Processing*, pages 18–19. Association for Computational Linguistics.
- Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2016. [Multilingual language processing from bytes](#). In *Proceedings of NAACL-HLT*, pages 1296–1306, San Diego, USA. Association for Computational Linguistics.
- Adrià de Gispert, Gonzalo Iglesias, and Bill Byrne. 2015. [Fast and accurate preordering for SMT using neural networks](#). In *Proceedings of NAACL*, pages 1012–1017.
- Song Han, Huizi Mao, and William J Dally. 2015. [Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding](#). *arXiv preprint arXiv:1510.00149*.
- Geoffrey E. Hinton. 2012. [A practical guide to training restricted Boltzmann machines](#). In *Neural Networks: Tricks of the Trade (2nd ed.)*, Lecture Notes in Computer Science, pages 599–619. Springer.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. [Long short-term memory](#). *Neural computation*, 9(8):1735–1780.

- Yoon Kim and Alexander M. Rush. 2016. [Sequence-level knowledge distillation](#). In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.
- Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2015. [Segmental recurrent neural networks](#). *CoRR*, abs/1511.06018.
- Terry Koo, Xavier Carreras, and Michael Collins. 2008. [Simple semi-supervised dependency parsing](#). In *Proceedings of ACL-08: HLT*, pages 595–603.
- Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernández Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. [Finding function in form: Compositional character models for open vocabulary word representation](#). In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530. Association for Computational Linguistics.
- Yijia Liu, Wanxiang Che, Jiang Guo, Bing Qin, and Ting Liu. 2016. [Exploring segment representations for neural segmentation models](#). In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 2880–2886.
- Shervin Malmasi, Marcos Zampieri, Nikola Ljubešić, Preslav Nakov, Ahmed Ali, and Jörg Tiedemann. 2016. [Discriminating between similar languages and Arabic dialect identification: A report on the third DSL shared task](#). In *Proceedings of the Third Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial3)*, pages 1–14, Osaka, Japan. The COLING 2016 Organizing Committee.
- Giovanni Molina, Fahad AlGhamdi, Mahmoud Ghoneim, Abdelati Hawwari, Nicolas Rey-Villamizar, Mona Diab, and Tamar Solorio. 2016. [Overview for the second shared task on language identification in code-switched data](#). In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 40–49, Austin, Texas. Association for Computational Linguistics.
- Vinod Nair and Geoffrey E. Hinton. 2010. [Rectified linear units improve restricted Boltzmann machines](#). In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814.
- Tetsuji Nakagawa. 2015. [Efficient top-down BTG parsing for machine translation preordering](#). In *Proceedings of ACL*, pages 208–218.
- Joakim Nivre. 2009. [Non-projective dependency parsing in expected linear time](#). In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359.
- Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. [Universal Dependencies v1: A Multilingual Treebank Collection](#). In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Association (ELRA).
- Wenzhe Pei, Tao Ge, and Baobao Chang. 2014. [Max-margin tensor neural network for Chinese word segmentation](#). In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 293–303, Baltimore, Maryland. Association for Computational Linguistics.
- Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. [Outrageously large neural networks: The sparsely-gated mixture-of-experts layer](#). *arXiv preprint arXiv:1701.06538*.
- Oscar Täckström, Ryan McDonald, and Jakob Uszkoreit. 2012. [Cross-lingual word clusters for direct transfer of linguistic structure](#). In *2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 477–487.
- David Talbot, Hideto Kazawa, Hiroshi Ichikawa, Jason Katz-Brown, Masakazu Seno, and Franz J. Och. 2011. [A lightweight evaluation framework for machine translation reordering](#). In *Proceedings of the Sixth Workshop on Statistical Machine Translation, WMT '11*, pages 12–21, Stroudsburg, PA, USA. Association for Computational Linguistics.
- David Talbot and John Talbot. 2008. [Bloom maps](#). In *Proceedings of the Meeting on Analytic Algorithms and Combinatorics*, pages 203–212. Society for Industrial and Applied Mathematics.
- Ivan Titov and James Henderson. 2007. [Fast and robust multilingual dependency parsing with a generative latent variable model](#). In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 947–951.
- Ivan Titov and James Henderson. 2010. [A latent variable model for generative dependency parsing](#). In Harry Bunt, Paola Merlo, and Joakim Nivre, editors, *Trends in Parsing Technology: Dependency Parsing, Domain Adaptation, and Deep Parsing*, pages 35–55. Springer Netherlands, Dordrecht.
- Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. [Word Representations: A Simple and General Method for Semi-supervised Learning](#). In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394, Stroudsburg, PA, USA. Association for Computational Linguistics.

- Jakob Uszkoreit and Thorsten Brants. 2008. [Distributed Word Clustering for Large Scale Class-Based Language Modeling in Machine Translation](#). In *ACL*, pages 755–762.
- David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. [Structured training for neural network transition-based parsing](#). In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 323–333.
- 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, ukasz 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, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. [Google’s neural machine translation system: Bridging the gap between human and machine translation](#). *CoRR*, abs/1609.08144.
- Fei Xia and Michael McCord. 2004. [Improving a statistical MT system with automatically learned rewrite patterns](#). In *Proceedings of COLING*, page 508.
- Jie Yang, Yue Zhang, and Fei Dong. 2017. [Neural word segmentation with rich pretraining](#). *CoRR*, abs/1704.08960.
- Meishan Zhang, Yue Zhang, and Guohong Fu. 2016. [Transition-based neural word segmentation](#). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 421–431, Berlin, Germany. Association for Computational Linguistics.
- Yue Zhang and Stephen Clark. 2007. [Chinese segmentation with a word-based perceptron algorithm](#). In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 840–847, Prague, Czech Republic. Association for Computational Linguistics.

Supplementary Material

A Quantization Details

The values comprising a generic embedding matrix $\mathbf{E} \in \mathbb{R}^{V \times D}$ are ordinarily stored with 32-bit floating-point precision in our implementation. For quantization, we first calculate a scale factor s_i for each embedding vector \mathbf{e}_i as

$$s_i = \frac{1}{b-1} \max_j |e_{ij}|.$$

Each weight e_{ij} is then quantized into an 8-bit integer as

$$q_{ij} = \lfloor \frac{1}{2} + \frac{e_{ij}}{s_i} + b \rfloor,$$

where the bias $b = 128$. Hence, the number of bits required to store the embedding matrix is reduced by a factor of 4, in exchange for storing the V additional scale values. At inference time, the embeddings are dequantized on-the-fly.

B FLOPs Calculation

The product of $\mathbf{A} \in \mathbb{R}^{P \times Q}$ and $\mathbf{b} \in \mathbb{R}^Q$ involves $P(2Q - 1)$ FLOPs, and our single ReLU hidden layer requires performing this operation once per timestep ($P=M$, $Q=H_0$). Here, H_0 denotes the size of the embedding vector \mathbf{h}_0 , which equals 408, 464 and 260 for our respective POS models as ordered in Table 2.

In contrast, each LSTM layer requires eight products per timestep, and the BTS model has four layers ($P=Q=320$). The particular sequence-to-sequence representation scheme of Gillick et al. (2016) requires at least four timesteps to produce a meaningful output: the individual input byte(s), and a start, length and label of the predicted span. A single timestep is therefore a relaxed lower bound on the number of FLOPs needed for BTS inference.

C Word Clusters

The word clusters we use are for the 250k most frequent words from a large unannotated corpus that was clustered into 256 classes using the distributed Exchange algorithm (Uszkoreit and Brants, 2008) and the procedure described in Appendix A of Täckström et al. (2012).

The space required to store them in a Bloom map is calculated using the formula derived by

Talbot and Talbot (2008): each entry requires $1.23 * (\log \frac{1}{\epsilon} + H)$ bits, where H is the entropy of the distribution on the set of values, and $\epsilon = 2^{-E}$, with E the number of error bits employed. We use 0 error bits and assume a uniform distribution for the 256 values, i.e. $H = 8$, hence we need 9.84 bits per entry, or 300KB for the 250k entries.

D Lang-ID Details

In our language identification evaluation, the 1,2,3,4-gram embedding vectors each have 6 or 16 dimensions, depending on the experimental setting. Their hashed vocabulary sizes (V_g) are 100, 1000, 5000, and 5000, respectively. The hidden layer size is fixed at $M=208$.

We preprocess data by removing non-alphabetic characters and pieces of markup text (i.e., anything located between $<$ and $>$, including the brackets). At test time, if this results in an empty string, we skip the markup removal, and if that still results in an empty string, we process the original string. This procedure is an artefact of the Wikipedia dataset, where some documents contain only punctuation or trivial HTML code, yet we must make predictions for them to render the results directly comparable to the literature.

E POS Details

The Small FF model in the comparison to BTS uses 2,3,4-grams and some byte unigrams (see feature templates in Table vii). The n -grams have embedding sizes of 16 and the byte unigrams get 4 dimensions. In our $\frac{1}{2}$ -dimension setting, the aforementioned dimensions are halved to 8 and 2.

Cluster features get embedding vectors of size 8. The hashed feature vocabularies for n -grams are 500, 200, and 4000, respectively. The hidden layer size is fixed at $M=320$.

bytes	$\forall i \in [0, 1], \forall j \in [0, 3] : l_{\pm i}^{\pm j}$
char n -grams	$\forall i \in [0, 3], \forall N \in [2, 4] : \{u_{\pm i}^{(N)}\}$
clusters	$\forall i \in [0, 3] : c_{\pm i}$

Table vii: **POS tagging feature templates.** i is a position relative to the focus token. l_j is the value of the j -th UTF8 byte from the start/end of a word. $\{u_{\pm i}^{(N)}\}$ designates the set of Unicode character n -grams in a word. c is the cluster id of a word.

char	$\forall i \in [0, 1] : \sigma_{\pm i} \cdot c; \quad \forall i \in [0, 2] \beta_{\pm i} \cdot c$
bigram	$\forall i \in [0, 1] : \sigma_{\pm i} \cdot b; \quad \beta_{\pm i} \cdot b$

Table viii: **Word segmentation feature templates.** ‘ $\beta_{\pm i}$ ’ denotes starting at the i -th character to the left/right of the front of the buffer. ‘c’ and ‘b’ denote character and character-bigram, respectively.

Features	Positions
char bigrams	for $i \in [0, 1] \sigma(i)_1$
	for $i \in [0, 2] \sigma(i)_{l_{\sigma(i)}}$
bytes	$\beta(0)_1$
	for $i \in [0, 1] \sigma(i)_1$
	for $i \in [0, 2] \sigma(i)_{l_{\sigma(i)}}$
has-swapped tags-main	$\beta(0)_1$
	for $i \in [0, 1] \sigma(i)$
	for $i \in [0, 1] \sigma(i)_1$
	for $i \in [0, 2] \sigma(i)_{l_{\sigma(i)}}$
tags-aux	$\beta(0)_1$
	for $i \in [0, 1] \sigma(i)_2 \sigma(i)_{l_{\sigma(i)-1}}$
	for $i \in [2, 3] \sigma(i)_1 \sigma(3)_{l_{\sigma(3)}}$
	$\beta(0)_2 \beta(0)_{l_{\beta(0)-1}} \beta_{l_{\beta(0)}}$
	for $j \in [1, 3] \beta(j)_1 \beta(j)_{l_{\beta(j)}}$

Table ix: **Preordering feature templates.** Each feature group applies to the set of positions given. $\sigma(i)$ denotes the i -th span from the top of the stack, and $\beta(j)$ the j -th span from the front of the buffer. Within a span s , the l_s tokens are $s_1 \dots s_{l_s}$, so s_1 is the leftmost token in s and s_{l_s} the rightmost.

<i>Model.</i>	<i>L.R.</i>	<i>Mom.</i>	γ	<i>Steps</i>	<i>D.P.</i>
C-64	0.03	0.8	32K	3.8M	0.2
C-256	0.03	0.8	32K	3.6M	0.4
C-64+B-04	0.03	0.8	64K	7.6M	0.3

Table x: **Segmentation:** Optimal hyperparameter settings per model for our segmentation experiments reported in Table 4. The columns show learning rate (L.R.), momentum factor (Mom.), the step-frequency at which the learning rate is scaled by 0.96 (γ), and the number of steps at which training was stopped because accuracy peaked on the held-out tuning data. The column *D.P.* shows the optimal dropout probability.

	<i>L.R.</i>	<i>Mom.</i>	γ	<i>Steps</i>
No POS tags	0.05	0.9	2k	38k
w/ POS tags	0.05	0.9	8k	46k
<i>POS model</i>	0.05	0.9	8k	500k
w/ tagger input fts.	0.1	0.8	4k	76k

Table xi: **Preordering:** Optimal hyperparameter settings obtained for our preordering experiments reported in Table 6. Columns have the same meanings as in Table x.

F Segmentation Details

Feature templates used in segmentation experiments are listed in Table viii. Besides, we define length feature to be the number of characters between top of σ and the front of β , this maximum feature value is clipped to 100. The length feature is used in all segmentation models, and the embedding dimension is set to 6. We set the cutoff for both character and character-bigrams to 2 in order to learn unknown character/bigram embeddings. The hidden layer size is fixed at $M=256$.

G Preordering Details

The feature templates for the preorderer look at the top four spans on the stack and the first four spans in the buffer; for each span, the feature templates look at up to the first two words and last two words within the span. The “vanilla” variant of the preorderer includes character n -grams, word bytes, and whether the span has ever participated in a SWAP transition. The POS features are the predicted tags for the words in these positions. Table ix shows the full feature templates for the preorderer.

#	Small FF 6 dim			Small FF 16 dim		
	<i>L.R.</i>	<i>Mom.</i>	γ	<i>L.R.</i>	<i>Mom.</i>	γ
0	0.4	0.9	8k	0.4	0.9	16k
1	0.4	0.9	32k	0.4	0.9	32k
2	0.4	0.9	32k	0.4	0.9	8k
3	0.3	0.9	64k	0.5	0.9	16k
4	0.4	0.8	100k	0.4	0.9	32k
5	0.5	0.8	100k	0.4	0.9	64k
6	0.3	0.9	32k	0.3	0.9	32k
7	0.3	0.9	100k	0.5	0.9	16k
8	0.4	0.9	32k	0.5	0.9	8k
9	0.4	0.9	32k	0.3	0.9	16k

Table xii: **Lang-ID:** Optimal hyperparameter settings obtained with the results reported in Table 1. The first column is the cross-validation fold, while the other columns have the same meanings as in Table x.

<i>Lang.</i>	<i>L.R.</i>	<i>Mom.</i>	γ	<i>Steps</i>	<i>Acc.</i>
Small FF					
bg	0.1	0.8	32k	90k	97.12
cs	0.05	0.9	32k	480k	97.97
da	0.05	0.9	32k	480k	94.17
en	0.01	0.9	128k	660k	92.50
fi	0.05	0.9	8k	210k	93.84
fr	0.1	0.8	64k	60k	95.10
de	0.1	0.8	8k	120k	91.23
el	0.08	0.9	64k	60k	96.88
id	0.08	0.8	32k	180k	91.60
it	0.08	0.8	128k	330k	96.79
fa	0.08	0.9	128k	60k	95.80
es	0.1	0.8	32k	60k	94.37
sv	0.1	0.9	8k	210k	94.54
Small FF + Clusters					
bg	0.08	0.8	64k	120k	97.72
cs	0.1	0.8	16k	420k	98.12
da	0.1	0.8	32k	360k	95.49
en	0.05	0.8	8k	510k	93.88
fi	0.1	0.8	8k	300k	94.97
fr	0.05	0.9	8k	630k	95.65
de	0.05	0.9	8k	480k	92.40
el	0.1	0.9	8k	60k	97.60
id	0.1	0.8	64k	150k	91.94
it	0.1	0.8	32k	270k	97.36
fa	0.08	0.9	64k	90k	96.24
es	0.05	0.9	128k	30k	95.01
sv	0.08	0.9	16k	150k	95.90
Small FF ($\frac{1}{2}$ Dim.) + Clusters					
bg	0.1	0.8	128k	210k	97.76
cs	0.05	0.9	32k	420k	98.06
da	0.05	0.9	16k	240k	95.33
en	0.05	0.8	8k	300k	93.06
fi	0.05	0.9	16k	390k	94.66
fr	0.08	0.9	128k	120k	95.28
de	0.08	0.9	16k	90k	92.13
el	0.08	0.9	16k	60k	97.42
id	0.08	0.9	8k	690k	92.15
it	0.05	0.9	64k	210k	97.42
fa	0.1	0.8	8k	510k	96.19
es	0.08	0.9	8k	60k	94.79
sv	0.1	0.8	16k	300k	95.76

Table xiii: **POS**: Optimal hyperparameter settings per language obtained for our POS experiments. Columns have the same meanings as in Table x. The final column shows the test set accuracies that back the averages shown in Table 2.