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 preordering 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 (preordering: §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., $E_{\text{bigrams}}$), with one embedding matrix $E_g \in \mathbb{R}^{V_g \times D_g}$ 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 $h_0 = [X_g E_g \mid \forall g]$. This cost is 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.

3. A single hidden layer, $h_1$, with $M$ rectified linear units (Nair and Hinton, 2010) is fully connected to $h_0$.

4. A softmax function models the probability of an output class $y$: $P(y) \propto \exp(\beta_y^T h_1 + b_y)$, 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 $v = \mathcal{H}(x) \mod V_g$, 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 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 preprocessing 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 $h_0$. 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 \( h_0 \) 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 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 \( \sim 15 \) 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-

---

\(^1\)github.com/google/cld3  
\(^2\)As of the date of this writing in 2017.  
\(^3\)For example, the commonly used English clusters from the BLLIP corpus is over 7 MB – people.csail.mit.edu/maestro/papers/bllip-clusters.gz

---

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

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.

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

Table 2: POS tagging. Embedded word clusters improves accuracy and allows the use of smaller embedding dimensions.
transitioning 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 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 alignment from Nakagawa (2015).

---

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

---

**Table 3: Segmentation Transition system.** Initially all characters are on the buffer \(β\) and the stack \(σ\) is empty: \([[]|c_1\ldots c_n]\). In the final state the buffer is empty and the stack contains the first character for each word.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPLIT (\left[\sigma, [i</td>
<td>β]\right] \rightarrow \left[\sigma</td>
</tr>
<tr>
<td>MERGE (\left[\sigma, [i</td>
<td>β]\right] \rightarrow \left[\sigma, [β]\right])</td>
</tr>
</tbody>
</table>

**Table 4: Segmentation results.** Explicit bigrams are useful.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang et al. (2016)</td>
<td>95.01</td>
<td>–</td>
</tr>
<tr>
<td>Zhang et al. (2016)-combo</td>
<td>95.95</td>
<td>–</td>
</tr>
<tr>
<td>Small FF, 64 dim</td>
<td>94.24</td>
<td>846KB</td>
</tr>
<tr>
<td>Small FF, 256 dim</td>
<td>94.16</td>
<td>3.2MB</td>
</tr>
<tr>
<td>Small FF, 64 dim, bigrams</td>
<td>95.18</td>
<td>2.0MB</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Transition</th>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND (\left([σ</td>
<td>i</td>
</tr>
<tr>
<td>SHIFT (\left([σ, [i</td>
<td>β]\right) \rightarrow \left([σ</td>
</tr>
<tr>
<td>SWAP (\left([σ</td>
<td>i</td>
</tr>
</tbody>
</table>
Table 6: Preordering results for English → Japanese. FRS (in $[0, 100]$) is the fuzzy reordering score (Talbot et al., 2011).

<table>
<thead>
<tr>
<th>Model</th>
<th>FRS</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nakagawa (2015)</td>
<td>81.6</td>
<td>-</td>
</tr>
<tr>
<td>Small FF</td>
<td>75.2</td>
<td>0.5MB</td>
</tr>
<tr>
<td>Small FF + POS tags</td>
<td>81.3</td>
<td>1.3MB</td>
</tr>
<tr>
<td>Small FF + Tagger input fts.</td>
<td>76.6</td>
<td>3.7MB</td>
</tr>
</tbody>
</table>

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


