SGNMT – A Flexible NMT Decoding Platform for Quick Prototyping of New Models and Search Strategies

Felix Stahlberg[†] and **Eva Hasler**[‡] and **Danielle Saunders**[†] and **Bill Byrne**[‡†]

[†]Department of Engineering, University of Cambridge, UK

[‡]SDL Research, Cambridge, UK

Abstract

This paper introduces SGNMT, our experimental platform for machine translation research. SGNMT provides a generic interface to neural and symbolic scoring modules (*predictors*) with left-to-right semantic such as translation models like NMT, language models, translation lattices, n-best lists or other kinds of scores and constraints. Predictors can be combined with other predictors to form complex decoding tasks. SGNMT implements a number of search strategies for traversing the space spanned by the predictors which are appropriate for different predictor constellations. Adding new predictors or decoding strategies is particularly easy, making it a very efficient tool for prototyping new research ideas. SGNMT is actively being used by students in the MPhil program in Machine Learning, Speech and Language Technology at the University of Cambridge for course work and theses, as well as for most of the research work in our group.

1 Introduction

We are developing an open source decoding framework called SGNMT, short for Syntactically Guided Neural Machine Translation.[1] The software package supports a number of well-known frameworks, including TensorFlow[2] (Abadi et al., 2016), OpenFST (Allauzen et al., 2007), Blocks/Theano (Bastien et al., 2012; van Merriënboer et al., 2015), and NPLM (Vaswani et al., 2013). The two central concepts in the SGNMT tool are *predictors* and *decoders*. Predictors are scoring modules which define scores over the target language vocabulary given the current internal predictor state, the history, the source sentence, and external side information. Scores from multiple, diverse predictors can be combined for use in decoding.

Decoders are search strategies which traverse the space spanned by the predictors. SGNMT provides implementations of common search tree traversal algorithms like beam search. Since decoders differ in runtime complexity and the kind of search errors they make, different decoders are appropriate for different predictor constellations.

The strict separation of scoring module and search strategy and the decoupling of scoring modules from each other makes SGNMT a very flexible decoding tool for neural and symbolic models which is applicable not only to machine translation. SGNMT is based on the OpenFST-based Cambridge SMT system (Allauzen et al., 2014). Although the system is less than a year old, we have found it to be very flexible and easy for new researchers to adopt. Our group has already integrated SGNMT into most of its research work.

We also find that SGNMT is very well-suited for teaching and student research projects. In the 2015-16 academic year, two students on the Cambridge MPhil in Machine Learning, Speech and Language Technology used SGNMT for their dissertation projects.[3] The first project involved using SGNMT with OpenFST for applying subword models in SMT (Gao, 2016). The second project developed automatic music composition by LSTMs where WFSAs were used to define the space of allowable chord progressions in 'Bach' chorales (Tomczak, 2016). The LSTM provides the 'creativity' and the WFSA enforces constraints

[1] http://ucam-smt.github.io/sgnmt/html/

[2] SGNMT relies on the TensorFlow fork available at https://github.com/ehasler/tensorflow

[3] http://www.mlsalt.eng.cam.ac.uk/Main/CurrentMPhils

Predictor	Predictor state	`initialize(·)`	`predict_next()`	`consume(token)`
NMT	State vector in the GRU or LSTM layer of the decoder network and current context vector.	Run encoder network to compute annotations.	Forward pass through the decoder to compute the posterior given the current decoder GRU or LSTM state and the context vector.	Feed back `token` to the NMT network and update the decoder state and the context vector.
FST	ID of the current node in the FST.	Load FST from the file system, set the predictor state to the FST start node.	Explore all outgoing edges of the current node and use arc weights as scores.	Traverse the outgoing edge from the current node labelled with `token` and update the predictor state to the target node.
n-gram	Current n-gram history	Set the current n-gram history to the begin-of-sentence symbol.	Return the LM scores for the current n-gram history.	Add `token` to the current n-gram history.
Word count	None	Empty	Return a cost of 1 for all tokens except `</s>`.	Empty
UNK count	Number of consumed UNK tokens.	Set UNK counter to 0, estimate the λ parameter of the Poisson distribution based on source sentence features.	For `</s>` use the log-probability of the current number of UNKs given λ. Use zero for all other tokens.	Increase internal counter by 1 if `token` is UNK.

Table 1: Predictor operations for the NMT, FST, n-gram LM, and counting modules.

that the chorales must obey. This second project in particular demonstrates the versatility of the approach. For the current, 2016-17 academic year, SGNMT is being used heavily in two courses.

2 Predictors

SGNMT consequently emphasizes flexibility and extensibility by providing a common interface to a wide range of constraints or models used in MT research. The concept facilitates quick prototyping of new research ideas. Our platform aims to minimize the effort required for implementation; decoding speed is secondary as optimized code for production systems can be produced once an idea has been proven successful in the SGNMT framework. In SGNMT, scores are assigned to partial hypotheses via one or many predictors. One predictor usually has a single responsibility as it represents a single model or type of constraint. Predictors need to implement the following methods:

- `initialize(src_sentence)` Initialize the predictor state using the source sentence.
- `get_state()` Get the internal predictor state.
- `set_state(state)` Set the internal predictor state.
- `predict_next()` Given the internal predictor state, produce the posterior over target tokens for the next position.

Predictor	**Description**
nmt	Attention-based neural machine translation following Bahdanau et al. (2015). Supports Blocks/Theano (Bastien et al., 2012; van Merriënboer et al., 2015) and TensorFlow (Abadi et al., 2016).
fst	Predictor for rescoring deterministic lattices (Stahlberg et al., 2016).
nfst	Predictor for rescoring non-deterministic lattices.
rtn	Rescoring recurrent transition networks (RTNs) as created by HiFST (Allauzen et al., 2014) with late expansion.
srilm	n-gram Kneser-Ney language model using the SRILM (Heafield et al., 2013; Stolcke et al., 2002) toolkit.
nplm	Neural n-gram language models based on NPLM (Vaswani et al., 2013).
rnnlm	Integrates RNN language models with TensorFlow as described by Zaremba et al. (2014).
forced	Forced decoding with a single reference.
forcedlst	n-best list rescoring.
bow	Restricts the search space to a bag of words with or without repetition (Hasler et al., 2017).
lrhiero	Experimental implementation of left-to-right Hiero (Siahbani et al., 2013) for small grammars.
wc	Number of words feature.
unkc	Applies a Poisson model for the number of UNKs in the output.
ngramc	Integrates external n-gram posteriors, e.g. for MBR-based NMT according Stahlberg et al. (2017).
length	Target sentence length model using simple source sentence features.

Table 2: Currently implemented predictors.

- `consume(token)` Update the internal predictor state by adding `token` to the current history.

The structure of the predictor state and the implementations of these methods differ substantially between predictors. Tab. 2 lists all predictors which are currently implemented. Tab. 1 summarizes the semantics of this interface for three very common predictors: the neural machine translation (NMT) predictor, the (deterministic) finite state transducer (FST) predictor for lattice rescoring, and the n-gram predictor for applying n-gram language models. We also included two examples (word count and UNK count) which do not have a natural left-to-right semantic but can still be represented as predictors.

2.1 Example Predictor Constellations

SGNMT allows combining any number of predictors and even multiple instances of the same predictor type. In case of multiple predictors we combine the predictor scores in a linear model. The following list illustrates that various interesting decoding tasks can be formulated as predictor combinations.

- `nmt`: A single NMT predictor represents pure NMT decoding.
- `nmt,nmt,nmt`: Using multiple NMT predictors is a natural way to represent ensemble decoding (Hansen and Salamon, 1990; Sutskever et al., 2014) in our framework.
- `fst,nmt`: NMT decoding constrained to an FST. This can be used for neural lattice rescoring (Stahlberg et al., 2016) or other kinds of constraints, for example in the context of source side simplification in MT (Hasler et al., 2016) or chord progressions in 'Bach' (Tomczak, 2016). The *fst* predictor can also be used to restrict the output of character-based or subword-unit-based NMT to a large word-level vocabulary encoded as FSA.
- `nmt,rnnlm,srilm,nplm`: Combining NMT with three kinds of language models: An RNNLM (Zaremba et al., 2014), a Kneser-Ney n-gram LM (Heafield et al., 2013; Stolcke et al., 2002), and a feedforward neural network LM (Vaswani et al., 2013).

Decoder	Description
greedy	Greedy decoding.
beam	Beam search as described in Bahdanau et al. (Bahdanau et al., 2015).
dfs	Depth-first search. Efficiently enumerates the complete search space, e.g. for exhaustive FST-based rescoring.
restarting	Similar to DFS but with better admissible pruning behaviour.
astar	A* search (Russell and Norvig, 2003). The heuristic function can be defined via predictors.
sepbeam	Associates hypotheses in the beam with only one predictor. Efficiently approximates system-level combination.
syncbeam	Beam search which compares hypotheses after consuming a special synchronization symbol rather than after each iteration.
bucket	Multiple beam search passes with small beam size. Can have better pruning behaviour than standard beam search.
vanilla	Fast beam search decoder for (ensembled) NMT. This implementation is similar to the decoder in Blocks (van Merriënboer et al., 2015) but can only be used for NMT as it bypasses the predictor framework.

Table 3: Currently implemented decoders.

- `nmt,ngramc,wc`: MBR-based NMT following Stahlberg et al. (2017) with n-gram posteriors extracted from an SMT lattice (`ngramc`) and a simple word penalty (`wc`).

3 Decoders

Decoders are algorithms to search for the highest scoring hypothesis. The list of predictors determines how (partial) hypotheses are scored by implementing the methods `initialize(·)`, `get_state()`, `set_state(·)`, `predict_next()`, and `consume(·)`. The *Decoder* class implements versions of these methods which apply to all predictors in the list. `initialize(·)` is always called prior to decoding a new sentence. Many popular search strategies can be described via the remaining methods `get_state()`, `set_state(·)`, `predict_next()`, and `consume(·)`. Algs. 1 and 2 show how to define greedy and beam decoding in this way.[4][5]

Tab. 3 contains a list of currently implemented decoders. The UML diagram in Fig. 1 illustrates the relation between decoders and predictors.

[4]Formally, `predict_next()` in Algs. 1 and 2 returns pairs of tokens and their costs.

[5]String concatenation is denoted with ·.

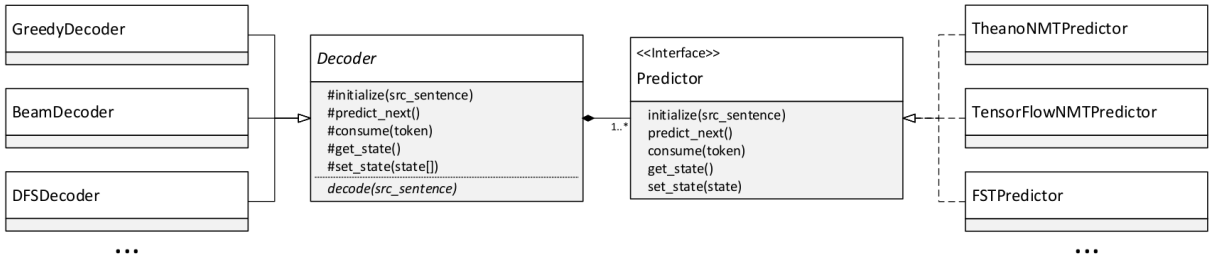

Figure 1: Reduced UML class diagram.

Algorithm 1 Greedy(src_sen)

```
initialize(src_sen)
h ← ⟨<s>⟩
repeat
    P ← predict_next()
    (t, c) ← arg max_{(t',c')∈P} c'
    h ← h · t
    consume(t)
until t = </s>
return h
```

NMT batch decoding The flexibility of the predictor framework comes with degradation in decoding time. SGNMT provides two ways of speeding up pure NMT decoding, especially on the GPU. The *vanilla* decoding strategy exposes the beam search implementation in Blocks (van Merriënboer et al., 2015) which processes all active hypotheses in the beam in parallel. We also implemented a beam decoder version which decodes multiple sentences at once (batch decoding) rather than in a sequential order. Batch decoding is potentially more efficient since larger batches can make better use of GPU parallelism. The key concepts of our batch decoder implementation are:

- We use a scheduler running on a separate CPU thread to construct large batches of computation (GPU jobs) from multiple sentences and feeding them to the *jobs* queue.
- The GPU is operated by a single thread which communicates with the CPU scheduler thread via queues containing jobs. This thread is only responsible for retrieving jobs in the *jobs* queue, computing them, and putting them in the *jobs_results* queue, minimizing the down-time of GPU computation.
- Yet another CPU thread is responsible for processing the results computed on the GPU in the *job_results* queue, e.g. by getting the n-best words from the posteriors. Processed jobs are sent back to the CPU scheduler where they are reassembled into new jobs.

Algorithm 2 Beam(n, src_sen)

```
initialize(src_sen)
H ← {(⟨<s>⟩, 0.0, get_state())}
repeat
    H_next ← ∅
    for all (h, c, s) ∈ H do
        set_state(s)
        P ← predict_next()
        H_next ← H_next ∪ ⋃_{(t',c')∈P} (h · t', c + c', s)
    end for
    H ← ∅
    for all (h, c, s) ∈ n-best(H_next) do
        set_state(s)
        consume(h_{|h|})
        H ← H ∪ {(h, c, get_state())}
    end for
until Best hypothesis in H ends with </s>
return Best hypothesis in H
```

This decoder is able to translate the WMT English-French test sets *news-test2012* to *news-test2014* on a Titan X GPU with 911.6 words per second with the word-based NMT model described in Stahlberg et al. (2016).[6] This decoding speed seems to be slightly faster than sequential decoding with high-performance NMT decoders like Marian-NMT (Junczys-Dowmunt et al., 2016) with reported decoding speeds of 865 words per second.[7] However, batch decoding with Marian-NMT is much faster reaching over 4,500 words

[6] Theano 0.9.0, cuDNN 5.1, Cuda 8 with CNMeM, Intel® Core i7-6700 CPU

[7] Note that the comparability is rather limited since even though we use the same beam size (5) and vocabulary sizes (30k), we use (a) a slightly slower GPU (Titan X vs. GTX

per second.[8] We think that these differences are mainly due to the limited multithreading support and performance in Python especially when using external libraries as opposed to the highly optimized C++ code in Marian-NMT. We did not push for even faster decoding as speed is not a major design goal of SGNMT. Note that batch decoding bypasses the predictor framework and can only be used for pure NMT decoding.

Ensembling with models at multiple tokenization levels SGNMT allows masking predictors with alternative sets of modelling units. The conversion between the tokenization schemes of different predictors is defined with FSTs. This makes it possible to decode by combining scores from both a subword-unit (BPE) based NMT (Sennrich et al., 2016) and a word-based NMT model with character-based NMT, masking the BPE-based and word-based NMT predictors with FSTs which transduce character sequences to BPE or word sequences. Masking is transparent to the decoding strategy as predictors are replaced by a special wrapper (*fsttok*) that uses the masking FST to translate `predict_next()` and `consume()` calls to (a series of) predictor calls with alternative tokens. The *syncbeam* variation of beam search compares competing hypotheses only after consuming a special word boundary symbol rather than after each token. This allows combining scores at the word level even when using models with multiple levels of tokenization. Joint decoding with different tokenization schemes has the potential of combining the benefits of the different schemes: character- and BPE-based models are able to address rare words, but word-based NMT can model long-range dependencies more efficiently.

System-level combination We showed in Sec. 2.1 how to formulate NMT ensembling as a set of NMT predictors. Ensembling averages the individual model scores in each decoding step. Alternatively, system-level combination decodes the entire sentence with each model separately, and selects the best scoring complete hypothesis over all models. In our experiments, system-level combination is not as effective as ensembling but still leads to moderate gains for pure NMT. However, a trivial implementation which selects the best translation in a postprocessing step after separate decoding runs is slow. The *sepbeam* decoding strategy reduces the runtime of system-level combination to the single system level. The strategy applies only one predictor rather than a linear combination of all predictors to expand a hypothesis. The single predictor is linked by the parent hypothesis. The initial stack in *sepbeam* contains hypotheses for each predictor (i.e. system) rather than only one as in normal beam search. We report a moderate gain of 0.5 BLEU over a single system on the Japanese-English ASPEC test set (Nakazawa et al., 2016) by combining three BPE-based NMT models from Stahlberg et al. (2017) using the *sepbeam* decoder.

Iterative beam search Normal beam search is difficult to use in a time-constrained setting since the runtime depends on the *target* sentence length which is a priori not known, and it is therefore hard to choose the right beam size beforehand. The *bucket* search algorithm sidesteps the problem of setting the beam size by repeatedly performing small beam search passes until a fixed computational budget is exhausted. Bucket search produces an initial hypothesis very quickly, and keeps the partial hypotheses for each length in buckets. Subsequent beam search passes refine the initial hypothesis by iteratively updating these buckets. Our initial experiments suggest that bucket search often performs on a similar level as standard beam search with the benefit of being able to support hard time constraints. Unlike beam search, bucket search lends itself to risk-free (i.e. admissible) pruning since all partial hypotheses worse than the current best complete hypothesis can be discarded.

4 Conclusion

This paper presented our SGNMT platform for prototyping new approaches to MT which involve both neural and symbolic models. SGNMT supports a number of different models and constraints via a common interface (*predictors*), and various search strategies (*decoders*). Furthermore, SGNMT focuses on minimizing the implementation effort for adding new predictors and decoders by decoupling scoring modules from each other and from the search algorithm. SGNMT is actively being used for teaching and research and we

1080), (b) a different training and test set, (c) a slightly different network architecture, and (d) words rather than subword units.

[8] https://marian-nmt.github.io/features/

welcome contributions to its development, for example by implementing new predictors for using models trained with other frameworks and tools.

Acknowledgments

This work was supported by the U.K. Engineering and Physical Sciences Research Council (EPSRC grant EP/L027623/1).

References

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Cyril Allauzen, Bill Byrne, Adrià de Gispert, Gonzalo Iglesias, and Michael Riley. 2014. Pushdown automata in statistical machine translation. *Computational Linguistics*, 40(3):687–723.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *International Conference on Implementation and Application of Automata*, pages 11–23. Springer.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. 2012. Theano: New features and speed improvements. In *NIPS*.

Jiameng Gao. 2016. Variable length word encodings for neural translation models. MPhil dissertation, University of Cambridge.

Lars Kai Hansen and Peter Salamon. 1990. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001.

Eva Hasler, Adrià de Gispert, Felix Stahlberg, Aurelien Waite, and Bill Byrne. 2016. Source sentence simplification for statistical machine translation. *Computer Speech & Language*.

Eva Hasler, Felix Stahlberg, Marcus Tomalin, Adrià de Gispert, and Bill Byrne. 2017. A comparison of neural models for word ordering. In *INLG*, Santiago de Compostela, Spain.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. Scalable modified Kneser-Ney language model estimation. In *ACL*, pages 690–696, Sofia, Bulgaria.

Marcin Junczys-Dowmunt, Tomasz Dwojak, and Hieu Hoang. 2016. Is neural machine translation ready for deployment? a case study on 30 translation directions. *arXiv preprint arXiv:1610.01108*.

Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. 2015. Blocks and fuel: Frameworks for deep learning. *arXiv preprint arXiv:1506.00619*.

Toshiaki Nakazawa, Manabu Yaguchi, Kiyotaka Uchimoto, Masao Utiyama, Eiichiro Sumita, Sadao Kurohashi, and Hitoshi Isahara. 2016. ASPEC: Asian scientific paper excerpt corpus. In *LREC*, pages 2204–2208, Portoroz, Slovenia.

Stuart J. Russell and Peter Norvig. 2003. *Artificial Intelligence: A Modern Approach*, 2 edition. Pearson Education.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *ACL*, pages 1715–1725, Berlin, Germany.

Maryam Siahbani, Baskaran Sankaran, and Anoop Sarkar. 2013. Efficient left-to-right hierarchical phrase-based translation with improved reordering. In *EMNLP*, pages 1089–1099, Seattle, Washington, USA.

Felix Stahlberg, Adrià de Gispert, Eva Hasler, and Bill Byrne. 2017. Neural machine translation by minimising the Bayes-risk with respect to syntactic translation lattices. In *EACL*, pages 362–368, Valencia, Spain.

Felix Stahlberg, Eva Hasler, Aurelien Waite, and Bill Byrne. 2016. Syntactically guided neural machine translation. In *ACL*, pages 299–305, Berlin, Germany.

Andreas Stolcke et al. 2002. SRILM – an extensible language modeling toolkit. In *Interspeech*, volume 2002, page 2002.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112. MIT Press.

Marcin Tomczak. 2016. Bachbot. MPhil dissertation, University of Cambridge.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. 2013. Decoding with large-scale neural language models improves translation. In *EMNLP*, pages 1387–1392, Seattle, Washington, USA.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.