mt@amazon

Sockeye:
Neural Machine Translation with MXNet

Amazon Core Machine Translation Team

github.com/awslabs/sockeye

Introduction

- Rapid evolution of Neural Machine Translation (NMT) methods

- NMT consistently outperforms previous approaches in international evaluations [WMT'16]

- As of today, no widely adopted NMT toolkit exists
  - Some are just research prototypes
  - Lack of scalability for training and inference
  - Nothing mature available for MXNet

Outline of this talk

1. Sequence-to-sequence modeling for NMT
2. Training and inference implementation
   + empirical evaluation
3. User intro for building NMT systems

What is Sockeye?

- Toolkit for sequence-to-sequence modeling in MXNet
  - Implements encoder-decoder models with attention [Bahdanau et al. 2014]
  - Supports different attention models [Luong et al. 2015]

- Named after the *Sockeye* salmon found in the Northern Pacific Ocean (Favorite fish around Seattle, WA)

- Open-sourced as Amazon’s official NMT framework

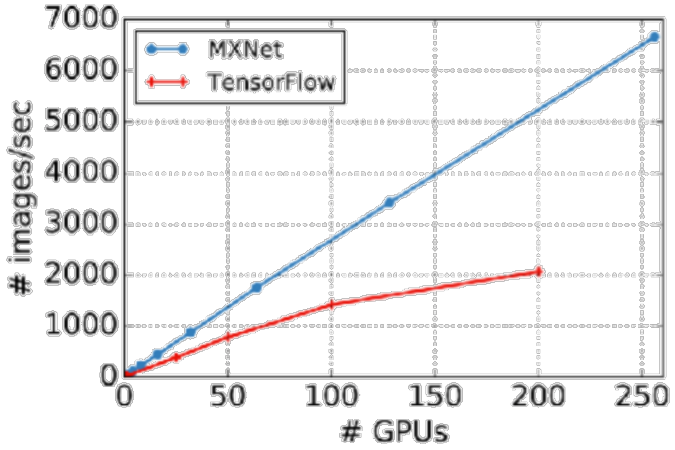
- Applicable to other sequence-to-sequence tasks, e.g.
  - Named Entity Recognition
  - Semantic Parsing
  - …

Why MXNet?

mt@

mxnet

- Fast & scalable
  - Native support for distributed training
  - Almost linear speedup with multiple GPUs



- Flexible programing model
  - Symbolic API (computation graphs)
  - Imperative API (NumPy on GPUs)

- Bindings for various languages (Python, C++, Scala, R, Julia, Perl)

- Officially supported by Amazon/AWS

Sequence-to-Sequence Modeling

An Intro to Sockeye's Neural Machine Translation Architecture

Sequence Model

Language model without Markov independence assumptions:

$$p(y) = \prod_{t=1}^{n} p(y_t | y_{1:t-1})$$

Recurrent Neural Network Language Model

1. Tokenize input sequence into discrete time steps

2. Embedding layer

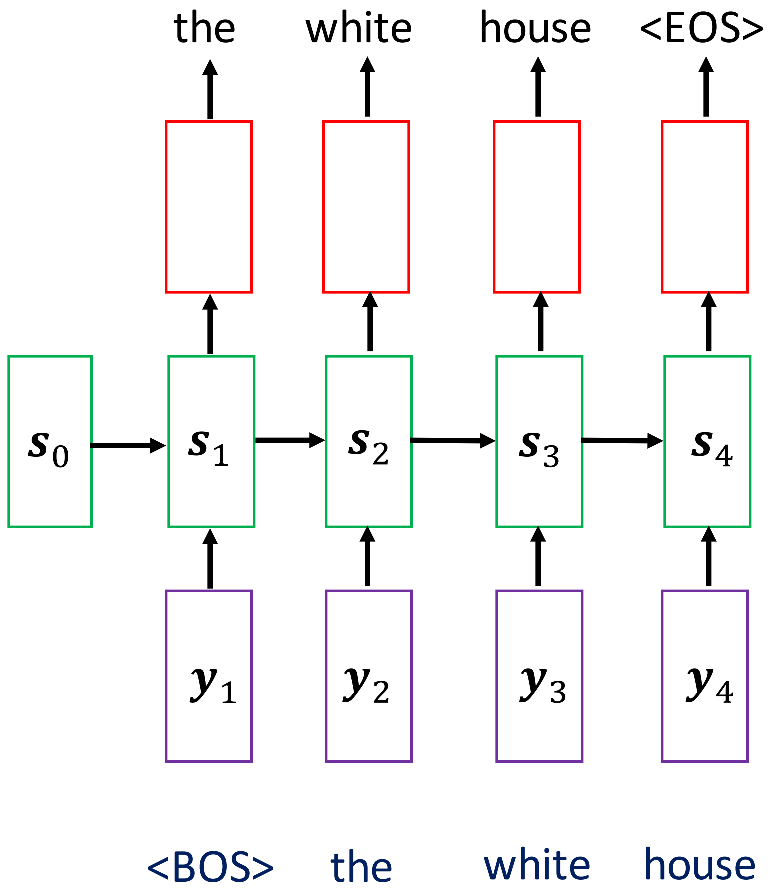
$$\boldsymbol{y}_t = \boldsymbol{W}_{\boldsymbol{E}} y_t\ ; \boldsymbol{W}_{\boldsymbol{E}} \in \mathbb{R}^{(|\boldsymbol{s}|,\,|V|)}, \boldsymbol{y}_t \in \mathbb{R}^{|\boldsymbol{s}|}$$

3. Recurrent hidden layer, e.g. RNN/LSTM/GRU

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, \boldsymbol{y}_t)\ ; f_{RNN} \coloneqq \tanh(\boldsymbol{U}\boldsymbol{s}_{t-1} + \boldsymbol{W}\boldsymbol{y}_t)$$

4. Output layer: predicts probability distribution over next words

$$p(\text{house}|\text{<BOS>}, \text{the}, \text{white}) = \text{softmax}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{s}_{\boldsymbol{3}} + \boldsymbol{b})$$



Sequence-to-Sequence Model

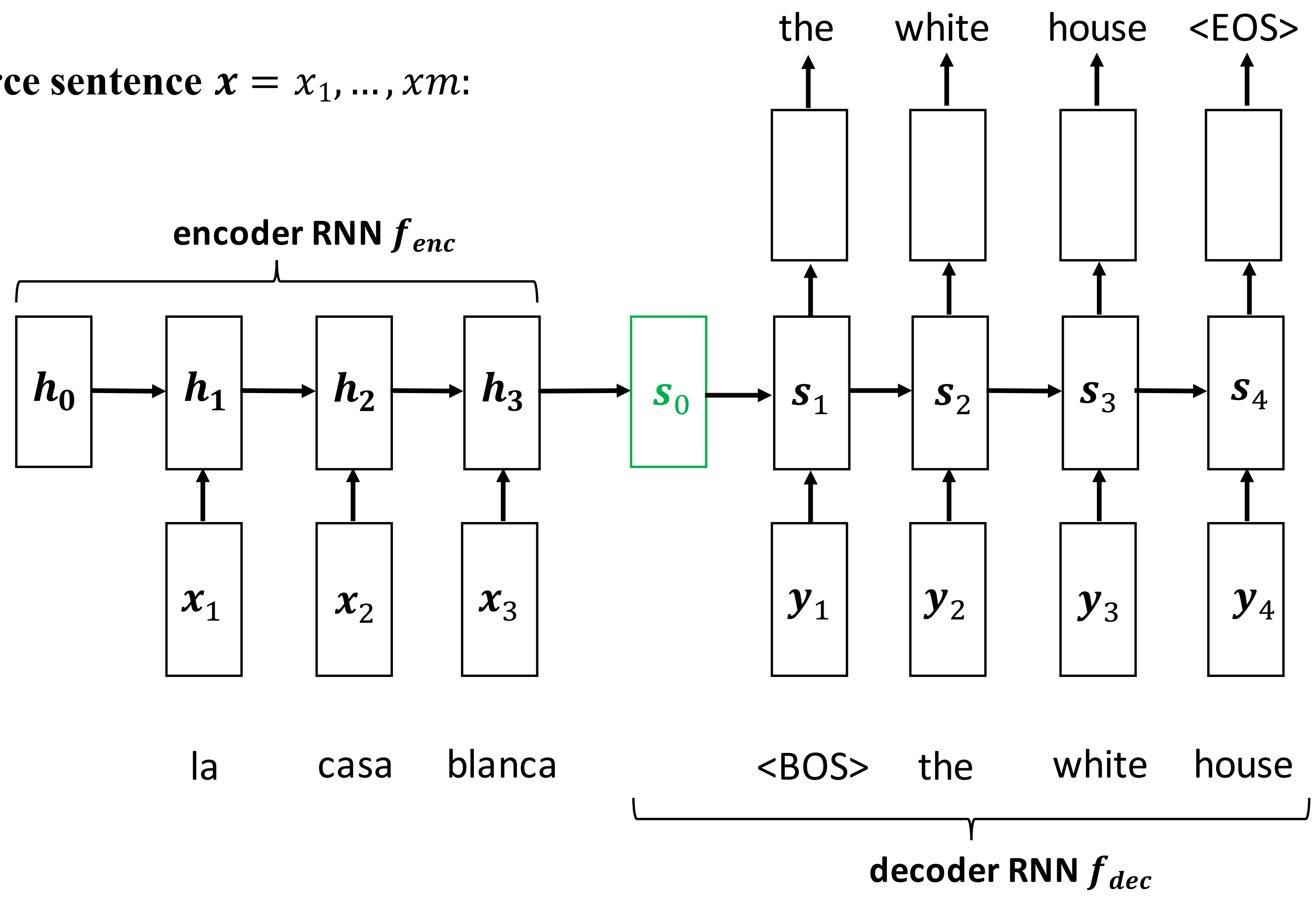
[Sutskever et al., 2014]

Language model **conditioned on source sentence** $\boldsymbol{x} = x_1, \dots, xm$:

$$p(y|x) = \prod_{t=1}^{n} p(y_t|y_{1:t-1}, x)$$

Encoded source sentence initializes decoder RNN:
$\boldsymbol{s}_0 = \tanh(\boldsymbol{W}_i \boldsymbol{h}_m + \boldsymbol{b}_i)$

Degrading translation quality for long sentences
Solution: **attention mechanism**



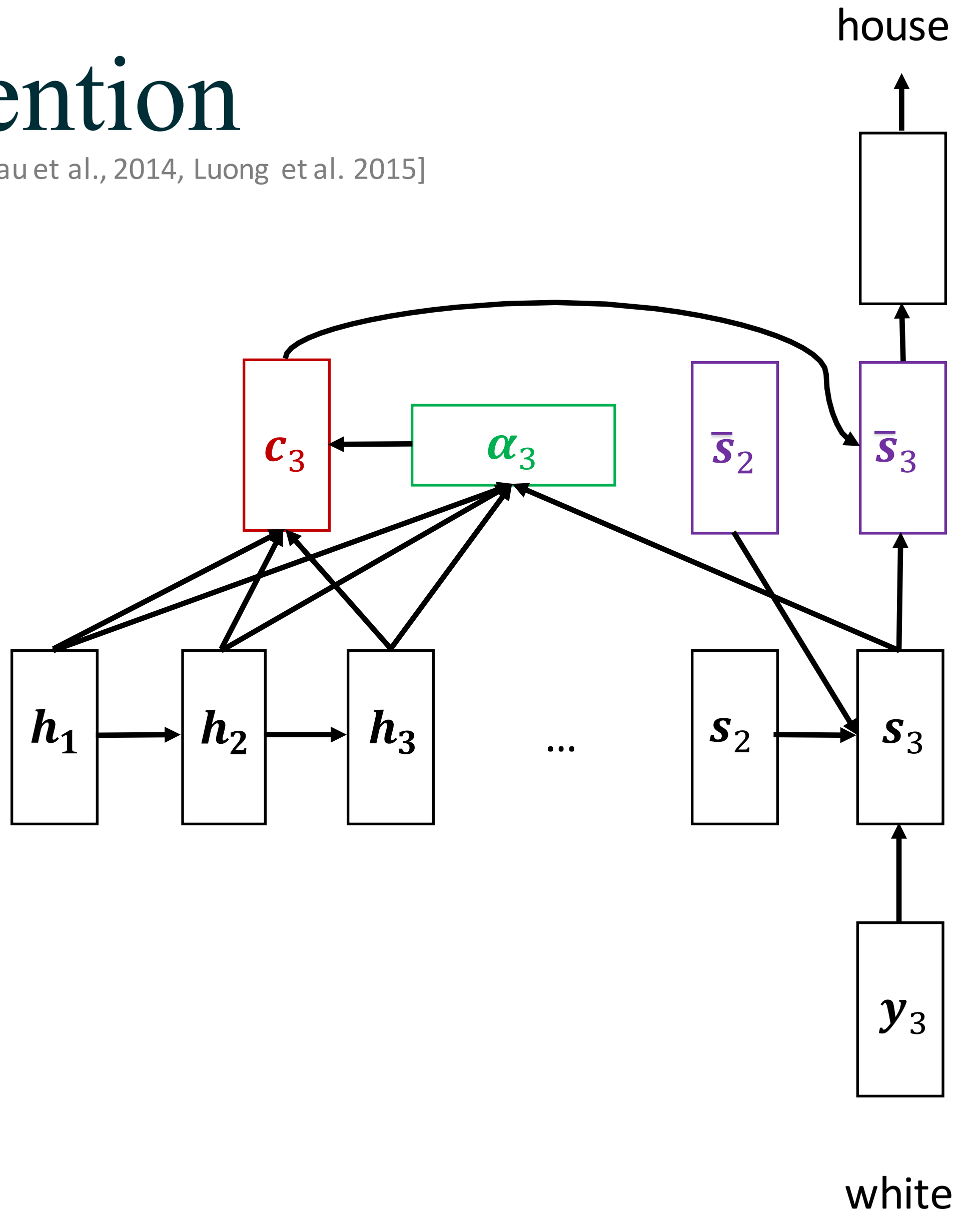
Sequence Decoding with Attention

[Bahdanau et al., 2014, Luong et al. 2015]

1. Decoder RNN additionally consumes **attentional vector $\bar{s}$**:
$$\boldsymbol{s}_t = fdec(\boldsymbol{s}_{t-1}, [\boldsymbol{y}_t, \bar{\boldsymbol{s}}_{t-1}])$$

2. Attentional vector $\bar{\boldsymbol{s}}$ combines previous RNN state $\boldsymbol{s}_t$ and **context vector $\boldsymbol{c}_t$**:
$$\bar{\boldsymbol{s}}_t = \tanh(\boldsymbol{W}_s[\boldsymbol{s}_t, \boldsymbol{c}_t])$$

3. Context vector $\boldsymbol{c}_t$ is a linear combination source states $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_m$:
$$\boldsymbol{c}_t = \sum_{i=1}^{m} \alpha_{ti} \boldsymbol{h}_i$$
$$\alpha_{ti} = softmax(score(\boldsymbol{s}_t, \boldsymbol{h}_i))$$



Attention Models in Sockeye

Name	$\text{score}(\mathbf{s}, \mathbf{h})$	Implemented in Sockeye
mlp [Bahdanau et al, 2014]	$\mathbf{v}_a^\top \tanh(\mathbf{W}_u \mathbf{s} + \mathbf{W}_v \mathbf{h})$	✓
concat [Luong et al. 2015]	$\mathbf{v}_a^\top [\mathbf{s}; \mathbf{h}]$	
dot [Luong et al. 2015]	$\mathbf{s}^\top \mathbf{h}$	✓
location [Luong et al. 2015]	$\mathbf{v}_{at}^\top \mathbf{s}_t$	✓
bilinear [Luong et al. 2015]	$\mathbf{s}_t^\top \mathbf{W}_a \mathbf{h}$	✓
coverage [Tu et al. 2015]	$\mathbf{v}_a^\top \tanh(\mathbf{W}_u \mathbf{s} + \mathbf{W}_v \mathbf{h} + \mathbf{W}_c \mathcal{C})$	✓

Training & Inference

High-Level Implementation Notes

MXNet Programming Models

Imperative

- NumPy like but GPU backend

```
from mxnet.ndarray import *



x = zeros((64, 12))
weights = zeros((128, 12))
x = FullyConnected(
        x, weights, num_hidden=128)
pred = SoftmaxActivation(x)
pred = pred.asnumpy()
```

Symbolic

- Optimized computation graph, auto-diff

```
from mxnet.symbol import *

y = Variable('y')
x = Variable('x')
weights = Variable('w')
x = FullyConnected(
        x, weights, num_hidden=128)
pred = SoftmaxOutput(x, y)
model = Module(pred)
model.fit(…)
model.forward_backward(data)
```

Training

Minimize cross-entropy loss with mini-batched SGD:

$$\mathcal{L}_\theta = \sum_{(x,y)\epsilon\mathbb{D}} -\log p(y|x;\theta)$$

- End-to-end training using MXNet's symbolic API
- Track perplexity (& BLEU) scores on held-out validation data
- Checkpoint every N updates
- Early-stopping if held-out scores do not improve for *n* checkpoints

Teacher forcing

Input to decoder RNN at time *t* is the correct target word at time *t*-1

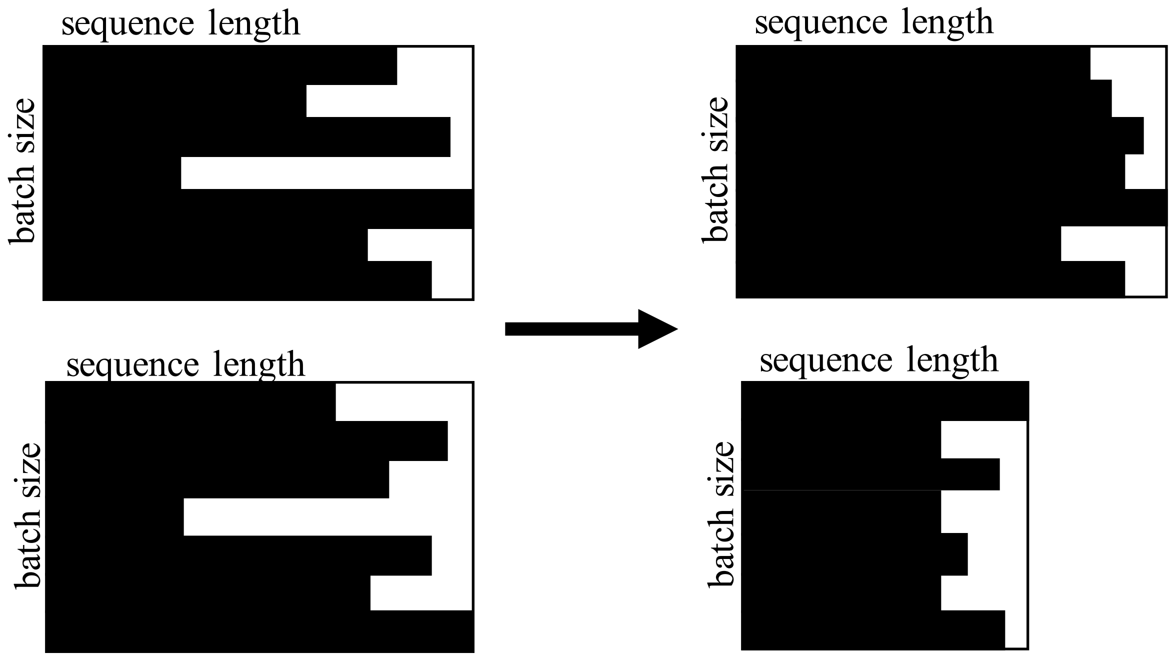
- Simplifies learning task
- Creates exposure bias

Training

No support for dynamic computation graphs in MXNet (landing soon!)
→ Unrolling of RNNs through time to a maximum sequence length
→ Padding of variable-length data

Bucketing

- Organize data into buckets of similar length
- Create one symbolic graph per bucket
  with shared memory and parameters
- Actual bucket key:
  (*source sequence length*, *target sequence length*)



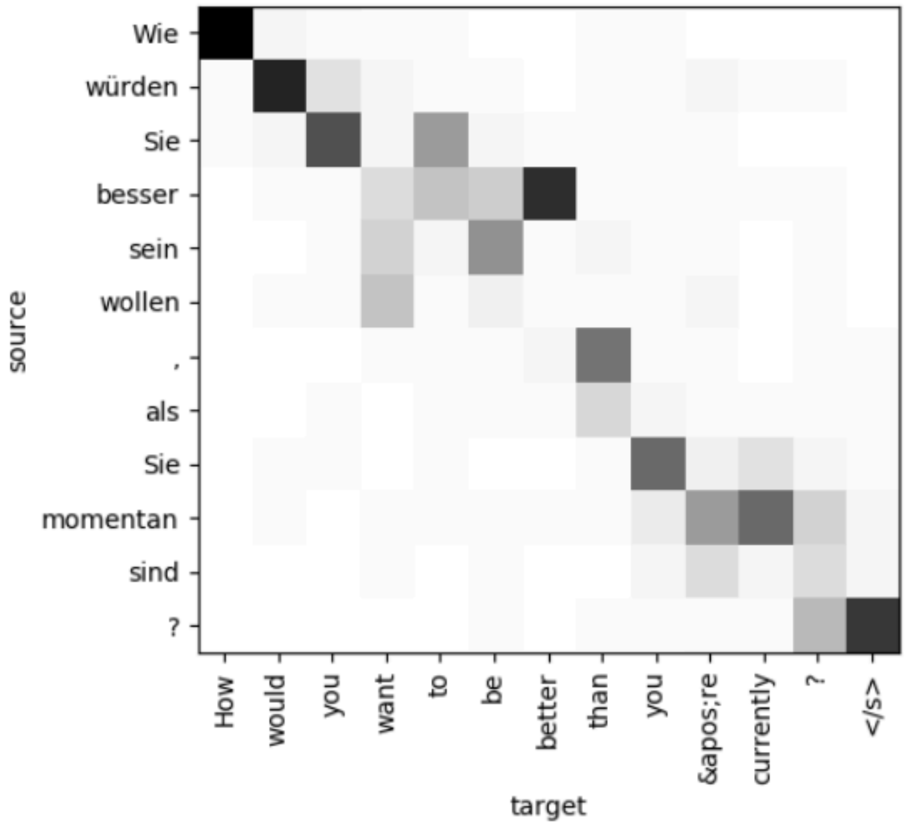
Inference

Beam search decoder

- Maintains & expands k-best hypotheses at each decoder time step until <EOS>
- Combines both MXNet programming models:
  - Symbolic: encode known source sequence
  - Imperative: iteratively generate target
- Supports ensembling of multiple models

Output visualization

Visualization of attention matrices (alignments)



Experiments

Translation Quality and Runtime Performance

Comparison vs Lamtram/DyNet

Baselines

1. Lamtram [Neubig, 2015], open-source NMT system based on DyNet [Neubig, 2017]

2. Phrase-based Statistical Machine Translation (PBMT) w/ Moses [Koehn, 2007]

Data

- IWSLT shared task data (train: 200k sentence pairs)
- German-English TED talk transcripts
- Preprocessing
    - Tokenization using Moses scripts
    - Sub-word segmentation with Byte-pair encoding (BPE) [Sennrich, 2015]

Systems

Chose system configurations similar to existing Lamtram baseline

- 1-layer LSTM encoder & decoder
- Dropout: 0.3
- ADAM optimizer

Sockeye-DOT	**Sockeye-MLP**	**Lamtram-MLP**
Dot attention	Mlp attention, 256 hidden units	• Dynamic batching: ~512 words per batch (~24 sentences) • “Gal” dropout [Gal & Grahramani, 2016]

Translation Results

System	BLEU ↑
PBSMT [no BPE]	28.7
Lamtram-MLP	31.5
Sockeye-DOT	28.3
Sockeye-MLP	32.4
Lamtram-MLP [ensemble of 3]	34.0
Sockeye-MLP [checkpoint average]	32.8
Sockeye-MLP [ensemble of 3]	**34.4**

IWSLT DE-EN test

- Meteor and TER metrics show the same behavior

Speed

Training speed
Single Tesla K80 (AWS p2 instance):

System	sec/epoch ↑	sent/sec ↑	total time (h) ↓
Sockeye-DOT	**1,490**	**132**	11.5
Sockeye-MLP	1,890	104	**5.7**
Lamtram-MLP	3,237	61	-

MXNet data parallelism
5.1x throughput with 6 GPUs

Decoding speed (beam size 5)
- Sockeye-MLP: ~4 sent/sec
- Lamtram-MLP: ~6 sent/sec

Memory Consumption

System	GPU memory consumption ↓
Sockeye-MLP (v1)	12 GB
Sockeye-MLP (v2: improved memory planning in MXNet)	9 GB
Sockeye-MLP (v3: re-designed MLP attention, computation graph optimization)	4.3 GB
Lamtram-MLP	approx. 2.8 GB

Running Sockeye

An Overview of Sockeye's CLI

Data Pre-Processing

Given raw input data:

```
The shares closed almost unchanged at 187.35 dollars.
The question comes alone: Collserola? Park or mountain?
```

Step 1 – Tokenize:

```
The shares closed almost unchanged at 187.35 dollars .
The question comes alone : Collserola ? Park or mountain ?
```

Step 2 – Sub-word encode:

```
The share _s closed a _lmost un _chang _ed at 18 _7 _. _35 dollar _s .
The question comes alone : Co _ll _s _er _ola ? Park or mountain ?
```

Sockeye CLI

Train with default settings:

```
python -m sockeye.train \
  --source train-corpus.de \
  --target train-corpus.en \
  --validation-source dev-corpus.de \
  --validation-target dev-corpus.en
  --output model-dir
```

Decode with default settings:

```
python -m sockeye.translate \
  --models model-dir
```

Model Options (1)

`--num-words`

Max vocabulary size (top N frequency-sorted)

`--word-min-count`

Min number of occurrences to be included in vocabulary

`--num-embed`

Size of word embeddings

`--rnn-num-hidden`

Size of hidden layer for encoder/decoder

`--rnn-num-layers`

Number of hidden layers in encoder/decoder

Model Options (2)

`--rnn-residual-connections`

Use residual connections for deep RNNs

`--attention-type`

Type of attention (dot, mlp, coverage)

`--weight-tying`

Share target embedding and output parameter weight matrix

`--attention-use-prev-word`

Feed previous target word embedding into attention

`--context-gating`

Use gate to adaptively weigh decoder input vs source context when decoding

Training Options

`--batch-size`

Mini-batch size for updates

`--optimized-metric`

Metric for early stopping (perplexity, BLEU)

`--dropout`

Dropout prob for word embeddings and RNNs

`--optimizer`

Learning algorithm (SGD, Adam)

`--loss`

Function to optimize (cross-entropy, smoothed cross-entropy)

Decoding Options

`--beam-size`

Size of the search beam

`--output-type`

Translation, translation with alignments, alignment plot

`--models`

One or more translation models (ensemble support)

Performance boost with parameter averaging:

Average parameters from the N best-scoring checkpoints:

```
python -m sockeye.average \
  --output model-dir/params.average \
  model-dir
```

mt@amazon

Sockeye: Neural Machine Translation with MXNet

Amazon Core Machine Translation Team

github.com/awslabs/sockeye