ニューラル機械翻訳における ミニバッチ構成法の違いによる影響の調査

森下 睦[†] 小田 悠介[†] Graham Neubig[††, †]
吉野 幸一郎[†, ‡] 須藤 克仁[‡‡] 中村 哲[†]

[†]奈良先端科学技術大学院大学 [††]Carnegie Mellon University
[‡]国立研究開発法人 科学技術振興機構 [‡‡]NTT コミュニケーション科学基礎研究所

{morishita.makoto.mb1, oda.yusuke.on9, koichiro, s-nakamura}@is.naist.jp
gneubig@cs.cmu.edu, sudoh.katsuhito@lab.ntt.co.jp

1 はじめに

ニューラル機械翻訳を学習する際には，複数のデータを一つのミニバッチとし，ミニバッチ単位で計算および学習を進めることが多い．一般に各ミニバッチの処理速度はミニバッチ内の最長文長に依存するため，事前に文長に従ってコーパスをソートすることで，処理が高速化することが知られている．しかし，これよって収束するまでにかかる実際の時間や，収束したモデルの精度がどのように変化するかについては検討されていない．また，現在公開されているツールキットは様々なソート手法やテクニックを用いてミニバッチを構成しており，これらのミニバッチ構成法の違いが学習にどのような影響を与えるかも調査されていない．

本研究では，様々なミニバッチ構成法を用いて実験を行い，それぞれが学習に与える影響を調査する．実験の結果，これまでに経験的に良いとされていたミニバッチ構成法は，学習の設定によっては逆に悪影響を与える可能性が示唆された．

2 ニューラル機械翻訳

近年ニューラルネットワークを利用した新しい機械翻訳手法であるニューラル機械翻訳 (Neural Machine Translation, NMT) が提案されている．初期の NMT は純粋な Encoder-Decoder モデル [12], [2] であったが，これに Attention と呼ばれる仕組みを持たせ，さらに精度を向上させた手法も提案されている [1], [7]. NMT はシンプルな手法でありながらも，様々な言語対で従来の統計的機械翻訳手法より高い翻訳精度を達成できることがわかっており [4]，機械翻訳手法の中で主流になりつつある．

2.1 ニューラル機械翻訳におけるミニバッチ

NMT はニューラルネットワークのモデル学習時に大量の行列計算を行う必要があり，学習に多くの時間がかかる．そのため，訓練データ全体を小さいサイズの集合 (ミニバッチ) に分け，各ミニバッチごとにまとめて計算を行うことで学習時間を短縮する手法が採用される．各ミニバッチに含まれるデータが固定長であれば，単純な行列計算を行うだけで学習が可能だが，NMT では異なる文長のデータを扱う．そこで短い文に対してはパディングを行う（文頭や文末に特定のトークンを付加してデータ長を合わせる）ことで，ミニバッチ内の最長の文に文長を合わせる．各ミニバッチを処理するためにかかる時間は，ミニバッチの文長に応じて長くなる．Sutskever ら [12] や Bahdanau ら [1] は，訓練データを文長に従ってソートした後ミニバッチを構成し，一つのミニバッチ中に似たような長さの文が多く含まれるようにすることで，各ミニバッチの処理時間が短くなると述べている．現在公開されている NMT ツールキットも，これらの報告を基に同様の処理を行うものが多い．

Algorithm 1 ミニバッチの構成法

```
C ← Training corpus
C ← sort(C) or shuffle(C)  ▷ コーパス全体をソートまたはシャッフル
B ← {}                     ▷ ミニバッチ
i ← 0, j ← 0
while i < C.size() do
    B[j] ← B[j] + C[i]
    if B[j].size() ≥ max minibatch size then
        B[j] ← padding(B[j])  ▷ ミニバッチ内の最大文長に合わせてパディング
        j ← j + 1
    end if
    i ← i + 1
end while
B ← shuffle(B)             ▷ 各ミニバッチの順序をシャッフル
```

しかし，このような手法は一つのミニバッチあたりにかかる処理時間は短くなるものの，モデルの収束にかかる時間に関しては言及されておらず，これに関する詳細な調査も行われていない．さらに，ツールキットによってはソート手法を含む細かな点でミニバッチの構成法が異なっており，これらの違いによる影響は不明である．

本研究ではこれらミニバッチ構成法の違いが NMT の学習に及ぼす影響について調査し，それぞれの手法における学習速度や，最終的に得られるモデルの精度の違いを確認する．

3 本研究で比較するミニバッチ構成法

まず，本研究で比較検討するミニバッチ構成法について詳しく述べる．アルゴリズム 1 にミニバッチを構成する擬似コードを示す．本研究では，本アルゴリズム中に含まれる設定を変化させその影響を調査する．比較する設定を以下に示す．

3.1 ミニバッチサイズ

本研究では，複数のミニバッチサイズを使用して学習を行い，その影響を調査する．

一般的に，ミニバッチサイズが大きければ各ミニバッチ間におけるパラメータの勾配の分散が小さくなるため，局所的な大きなパラメータの変化が起こりにくく，より安定した収束が期待できる．また，一度にまとめて処理する文数が大きくなるため，並列計算に特化したデバイスを使用した場合，単位時間あたりの学習文数が増える．しかし，ミニバッチサイズを大きくすると，それに伴いより大きなメモリが必要となる．

3.2 ミニバッチサイズの単位

本研究では，ミニバッチサイズの単位を文数または目的言語文の単語数とし，その違いを検討する．

一般的な NMT ツールキットは，各ミニバッチに含まれる文数を制御することでミニバッチサイズを決定することが多い．しかし，ツールキットによっては各ミニバッチに含まれる目的言語文の単語数を基にミニバッチサイズを決定しているものもある [1]．

NMT では，各目的言語文の単語に対して正解単語とのロスを計算し，これらを基にパラメータのアップデートを行う．そのため，文数を基にミニバッチを決定すると各ミニバッチに含まれる目的言語文の単語数が一定ではなくなり，ロスが計算される回数が一定でなくなる．文数ではなく目的言語文の単語数を基にミニバッチサイズを決定することで，各ミニバッチ間でのロスが計算される回数が一定となり，この問題を解消できる．

3.3 コーパスのソート手法

一般にオンライン学習においてはデータをシャッフルして学習するが，NMT では Sutskever ら [12] や Bahdanau ら [1] の報告に基づき，事前にコーパスをソートして学習する場合が多い．本研究では以下の 5 つのソート手法を比較する．

- コーパス全体をシャッフル (以下，shuffle)
- 原言語文長を基にソート (以下，src)
- 目的言語文長を基にソート (以下，trg)
- 原言語文長を基にソートし，同一文長のものについては目的言語文長を基にソート (以下，src_trg)
- 目的言語文長を基にソートし，同一文長のものについては原言語文長を基にソート (以下，trg_src)

src では，エンコーダ側が処理する単語数が減り，パディングされたトークンの Attention の重みが誤って大きくなることを防げる防ぐことができる．また，trg ではデコーダ側が処理する単語数が減り，ロスの計算回数が減る利点がある．Luong ら [7] や Bahdanau ら [1] のモデルではデコーダ側の処理の方が計算量が大きいため，trg や trg_src を採用するツールキットが多い [2]．

4 実験

4.1 実験設定

各ミニバッチ構成法の影響を調査するために，様々な条件の基で比較実験を行った．各手法で共通の実験設定は以下の通りである．

NMT モデルは Luong ら [7] によって提案された Global Attention と Attentional Feeding を組み合わせたモデルを用い，これに加えて，エンコーダは Bahdanau ら [1] の手法で用いられた Bidirectional Encoder を使用した．Encoder, Decoder の LSTM は 1 層とし，Hidden 層, Embed 層のユニット数はそれぞれ 512，語彙数は原言語，目的言語共に train セット中の頻出単語 65536 語とした．各層間では Dropout を行い，その確率は 30%とした．学習時に使用する乱数を固定することで，パラメータの初期値は全ての実験で同一となるようにした．学習アルゴリズムには Adam [5]($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$) または SGD(学習率 $\eta = 0.1$) を使用した．各ミニバッチ中でパディングが必要になった場合，末尾に文末を表すトークンを一定数付加することで，文長がミニバッチ中の最長文長と同一になるようにした．コーパスは，日本語側は KyTea [10] を用いて単語分割を行い，英語側は Moses に付属する tokenizer.perl[3] を使用して単語分割を行った後，小文字化を行った．NMT の学習には ASPEC-JE [8] の train セットのうち上位 200 万文 [4] を用い，英日翻訳モデルを学習した．抽出された 200 万文のうち，原言語文または目的言語文の文長が 60 単語を超えている文についてはコーパスから除外した．test セット，dev セットについても，同様に ASPEC-JE を使用した．NMT ツールキットには DyNet [9] を用いて構築された NMTKit[5] を用いた．train セットを 4 万文を学習するごとに dev セット，test セットを翻訳し，それぞれの Log Perplexity および BLEU スコア [11] [6] を記録した．

表 1 に本研究で比較調査する手法を示す [7]．それぞ

[1] この手法を採用しているツールキットとして lamtram (`https://github.com/neubig/lamtram`) がある．

[2] src を使用しているツールキットとして OpenNMT [6] がある．trg を使用しているツールキットとして Nematus (`https://github.com/rsennrich/nematus`), KNMT [3] がある．trg_src を使用しているツールキットとして lamtram (`https://github.com/neubig/lamtram`) がある．

[3] `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl`

[4] ASPEC-JE は自動で文アライメントが取られた対訳コーパスであり，アライメントの信頼度順にソートされている．train セット全体は約 300 万文あるが，下位の文についてはノイズを多く含んでいるため，上位 200 万文のみを使用した．

[5] `https://github.com/odashi/nmtkit`

[6] 本研究で計測した BLEU スコアは，参照訳中に未知語が含まれている場合それらを未知語トークンに置き換えた上で評価した．そのため，本来の BLEU スコアと比較して多少の誤差が含まれている．

[7] 手法 (e) では，コーパス内の平均目的語文長 $\times$ 64 $\simeq$ 2055 となるため，バッチサイズを 2055 words とした．

表 1: 比較手法

	ミニバッチ構成法	学習アルゴリズム
(a)	64 sentences	Adam
(b)	32 sentences	Adam
(c)	16 sentences	Adam
(d)	8 sentences	Adam
(e)	2055 words	Adam
(f)	64 sentences	SGD

表 2: 手法 (a) においてコーパス全体を 1 回学習するために必要な平均時間

ソート手法	平均必要時間 (hour)
shuffle	8.08
src	6.45
trg	5.21
src_trg	4.35
trg_src	4.30

れの手法において，3.3 節で述べたソート手法を全て比較する．

4.2 実験結果，考察

図 1, 2 に Test セットにおける Log Perplexity および BLEU スコアの変化を示す．また，表 2 に手法 (a) においてコーパス全体を 1 回学習するために必要な平均時間を示す．

4.2.1 ミニバッチサイズの違いによる影響

手法 (a), (b), (c), (d) の実験結果より，学習アルゴリズムとして Adam を用いた場合，ミニバッチサイズが最終的な精度および Perplexity に影響することが示唆された．これは 3.1 節でも述べたように，ミニバッチサイズを大きくすることでパラメータ更新量の分散が小さくなることが影響していると考えられる．今回の実験結果だけから最適なミニバッチサイズを求めることは困難だが，メモリの上限までミニバッチサイズを大きくすることで，処理速度，収束速度および精度の面で，良い結果が得られる可能性が示唆された．

4.2.2 ミニバッチサイズの単位の違いによる影響

手法 (a), (e) の実験結果に着目すると，手法 (a) では shuffle，手法 (e) では src を使用した場合に，Perplexity が速く減少することが確認できた．

我々は，ロスの計算回数が異なることで，Adam [5] などの Momentum を利用する学習アルゴリズムを使用する際に特に大きな影響を与えると考えていた．しかし，これら 2 つの Perplexity の減少速度に大きな差は無く，ミニバッチサイズの単位については速度及び精度には大きな影響を及ぼさないと考えられる．

4.2.3 Adam を用いた場合のソート手法の違いによる影響

手法 (a), (b), (c), (d), (e) の実験結果に共通して，ソート手法として shuffle または src を用いた場合，他のソート手法と比べてミニバッチサイズによらず安定して Perplexity が減少していることがわかる．これは，trg でソートすることにより目的言語文のパディング回数が減り，それに応じて文末を表すトークンの出現数が減ったことが原因でないかと考えられる．学習回数が減るため，他のソート手法と比較して文末を表すトークンの推定精度が落ち，これが Perplexity および精度に影響した可能性がある．

また，文長が類似している文はその文の特徴も類似している可能性がある [8]．その場合，文長に従ってソートすることで一つのミニバッチ内に類似した特徴を持った文が多く集まるため，局所解に収束しやすくなっている可能性がある．

表 2 より，現在多くのツールキットで使われている trg および trg_src は処理速度については他の手法と比べて高速である．しかし，収束速度や精度の側面から見ると shuffle や src を用いたほうがより良い可能性が示唆された．

4.2.4 SGD を用いた場合のソート手法の違いによる影響

手法 (a), (f) を比較すると，Adam を用いた場合はソート手法によってモデルの収束速度および精度が大きく違うが，SGD を用いた場合はいずれのソート手法を用いた場合でも Perplexity の下がり方は同様である．このことから，SGD を用いる場合は処理速度が高速な trg_src を使用すると良いことがわかった．

Wu らにより，学習初期には Adam を用い，その後 SGD に移行することで高速かつ高精度に学習を進める手法が提案されている [13]．今回の実験結果から，このような手法を用いる場合，Adam を使用している間は shuffle または src を使用し，SGD に移行したあとはミニバッチあたりの処理速度が高速な trg_src を用いると，さらに効率的に学習が進められると考えられる．

5 おわりに

本研究では，NMT の学習においてミニバッチの構成法が学習に与える影響について調査を行った．実験結果より，NMT を学習する際にはミニバッチの構成法についても考慮することで，より高速かつ高精度なシステムが構築できる可能性が示唆された．

また実験結果より，ミニバッチサイズは処理速度だけでは無く， 最終的な Perplexity や精度についても影響することがわかった．Adam を使用する場合は shuffle または src の方が，一般的に幅広く用いられている trg より安定して Perplexity が減少することが示唆された．SGD を使用する場合は，いずれのソート手法を用いた場合でも Perplexity の下がり方は一様であるため，最も処理速度が早い trg_src を使用すると良いと考えられる．

今後の課題としては，より汎用的な知見を得るために他のコーパス，言語対，学習アルゴリズムを含むより様々な条件の基で実験を行い，結果を確認することなどが挙げられる．

謝辞 本研究の一部は JSPS 科研費 JP16H05873, JP24240032 の助成を受けたものです．

[8] 今回使用したコーパスの場合，文長が短い文は名詞句のみを含んだ文が多いなどの特徴を目視により確認した．

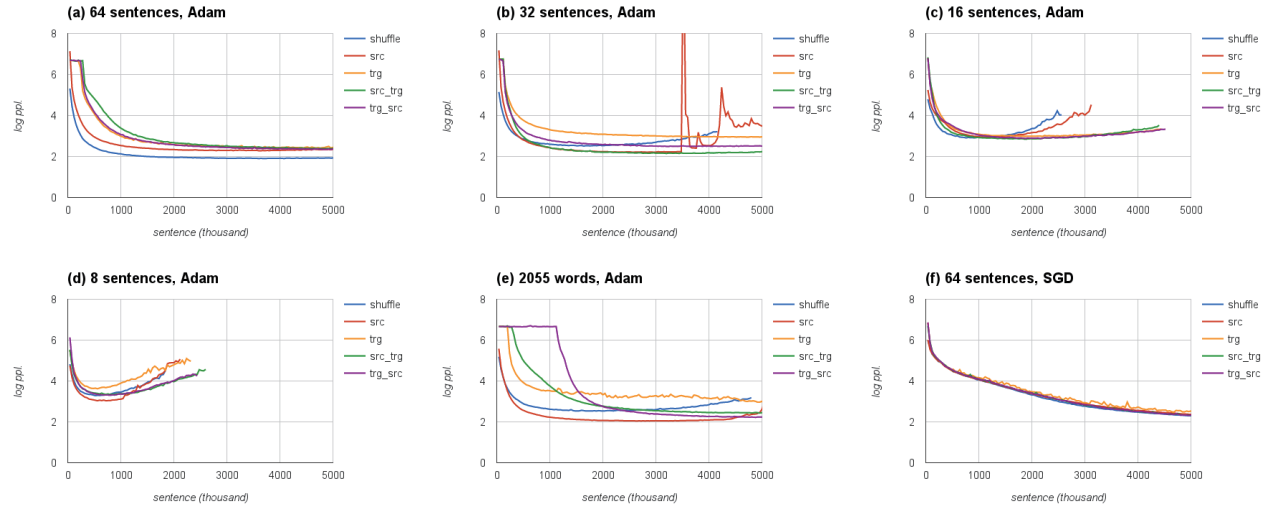

図 1: Test セットにおける Log Perplexity の変化

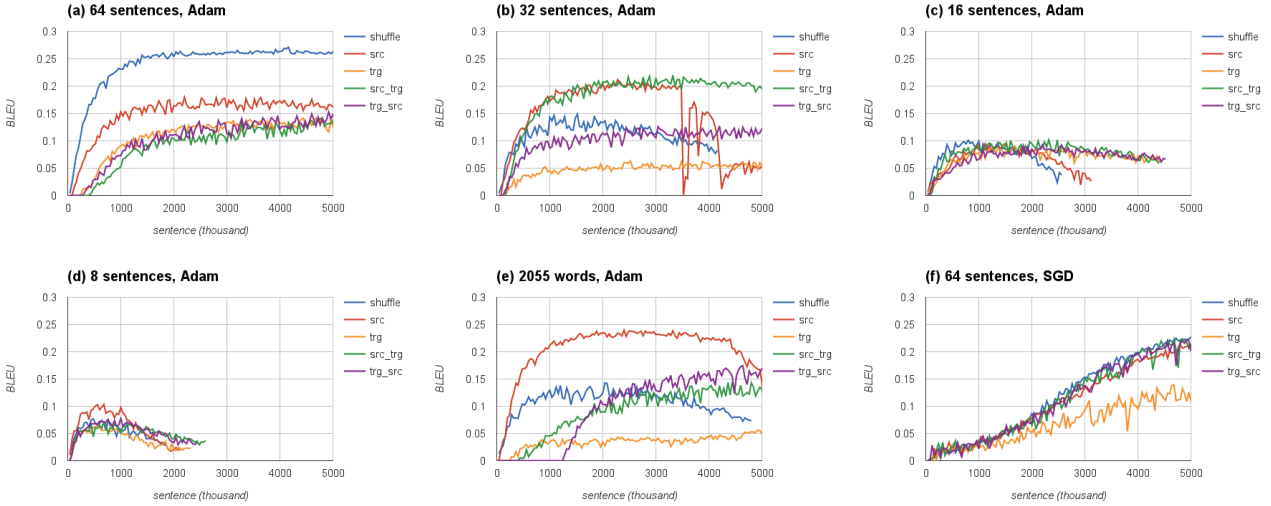

図 2: Test セットにおける BLEU スコアの変化

参考文献

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

[2] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of EMNLP*, pp. 1724–1734, 2014.

[3] Fabiën Cromieres. Kyoto-nmt: a neural machine translation implementation in chainer. In *Proceedings of COLING*, 2016.

[4] Marcin Junczys-Dowmunt, Tomasz Dwojak, and Hieu Hoang. Is neural machine translation ready for deployment? a case study on 30 translation directions. In *Proceedings of IWSLT*, 2016.

[5] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of ICLR*, 2015.

[6] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*, 2017.

[7] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP*, pp. 1412–1421, 2015.

[8] Toshiaki Nakazawa, Manabu Yaguchi, Kiyotaka Uchimoto, Masao Utiyama, Eiichiro Sumita, Sadao Kurohashi, and Hitoshi Isahara. Aspec: Asian scientific paper excerpt corpus. In *Proceedings of LREC*, 2016.

[9] Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

[10] Graham Neubig, Yosuke Nakata, and Shinsuke Mori. Pointwise prediction for robust, adaptable Japanese morphological analysis. In *Proceedings of ACL*, pp. 529–533, 2011.

[11] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of ACL*, pp. 311–318, 2002.

[12] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of NIPS*, pp. 3104–3112, 2014.

[13] 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, Łukasz 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. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.