DEIM2020 P1-32(day1 p97)

点群データベースに対する類似度検索の検討

小出　智士†　　河野　圭祐†　　沓名　拓郎†

† 株式会社 豊田中央研究所　〒 480–1192 愛知県長久手市横道 41–1
E-mail: †{koide,kawano,kutsuna}@mosk.tytlabs.co.jp

あらまし　LiDAR などのセンサーから収集される大規模な 3 次元点群データ処理は自動運転技術で重要な役割をもつ重要なデータ型である．本研究では複数の点群データから構成される点群データベースから，ユーザが指定した問い合わせ点群に類似した点群を検索する，という問題の検討を行う．我々は分布間の類似度を定める Wasserstein 距離に着目する．Wasserstein 距離の妥当性を確認するために，LiDAR によって得られた大規模かつオープンな点群データベース (KITTI データセット) を用いて検証を行った．その妥当性や課題を検討した結果を報告する．

キーワード　点群データベース, 類似度検索, Wasserstein 距離

1　は じ め に

近年，LiDAR などセンサーから収集される 3 次元点群データの処理技術が注目されている．この一つの背景として，自動運転技術における LiDAR を用いた環境センシング，および認識の重要性が増していることが挙げられる．このような環境認識のための認識モデルは大規模な点群データの履歴および点群深層学習 (例えば [1], [2]) によって，近年，性能向上が図られている．それに伴って大規模な点群データのマネジメントも重要な課題になっている．

点群データの表現としては様々な分類が考えられるが，自動運転のセンシングの文脈では以下の 2 タイプが想定される．

単一点群型　巨大な 1 つの点群を格納する．例えば井川ら [3] は名古屋大学構内をおよそ 7800 万点の点群で表現し，ある地点の周囲にある点群を検索する技術を検討した．

複数点群型　複数の点群を格納する．例えば KITTI データセット [4] には 1 シーンあたり 10 万点を超える点群が 40000 シーン以上格納されている．

単一点群型は高精度地図などを表現するのに適しているのに対し，複数点群型は動的な道路環境を表現するのに向いており，目的に沿った使い分けが重要である．以降では複数点群型に焦点を絞って検討を行うこととし，単に「点群データベース」もしくは「点群データ」と言った場合には複数点群型を想定するものとする．

我々の研究の目的は，複数の点群を格納した大規模データベースに対して類似度検索を実現することである．想定される類似度検索のユースケースとしては例えば以下のようなものがあるだろう．

> 点群から歩行者認識を行う，という認識モデルを機械学習ベースで開発している．あるテストケースの点群 X で認識ミスが発生した．X と類似したシーンを (必ずしも教師データが付与されていない) 大規模点群データベースから検索し，追加的な検証を行いたい．

ここで問題となるのは「類似度としてどのような尺度を用いるべきか」という点である．点群データは通常のテーブル型のデータとは異なる以下のような性質があるため，類似度を定義を考える際には以下の性質を反映したものでなければならない．

（1）　**点群データは可変サイズのデータ型である**．すなわち各シーンに含まれる点の数は同じセンサデバイスを用いた場合でも異なることが一般的である．

（2）　**点群データは順序データではない**．すなわちあるシーンに N 個の点 $(x_1, \cdots, x_N)$ が含まれるとする．この順序を適当な置換 π を用いてシャッフルして新しいデータ $(x_{\pi(1)}, \cdots, x_{\pi(N)})$ を作った場合でもデータが持つ意味は変わらない．

本研究では，これらの性質を考慮可能な点群間の距離関数として Wasserstein 距離 [5] を検討する．2 章で Wasserstein 距離を導入したあと，3 章では Wasserstein 距離を用いる妥当性について検証する．4 章では高速な類似度検索アルゴリズムを提案し，5 章で実データを用いた評価を行う．

2　準　　備

2.1　点群データの表現

点群はデルタ関数の重ね合わせ (混合分布) として書くことができるため，Wasserstein 距離が利用可能である．具体的には，点群 $A = \{x_1, \cdots, x_N\}$ はデルタ関数 $\delta(x)$ を用いて

$$A(x) = \frac{1}{N} \sum_{i=1}^{N} \delta(x - x_i) \tag{1}$$

と表現できる．ここで $x_i \in \mathbb{R}^3$ である．より一般的には $\sum_{i=1}^{N} a_i = 1$ を満たす重みベクトル $\mathbf{a}$ を用いて

$$A(x) = \sum_{i=1}^{N} a_i \delta(x - x_i) \tag{2}$$

とできるので，以降ではこの表現を採用する．点群 $B = \{y_1, \cdots, y_M\}$ は重みベクトル $\mathbf{b}$ を持つものとする．また，ベクトルもしくは行列の成分ごとの積の和 (内積) を $\langle \cdot, \cdot \rangle$ で表す．

2.2 Wasserstein 距離

前節では点群 A, B を確率測度として表現した．Wasserstein 距離 $W(A, B)$ は一般には確率測度間の距離関数を定めるものである．確率測度がデルタ関数の混合で書ける場合は，以下のように (一般のケースよりも) わかりやすい形で書くことができる．

$$W^2(A, B) = \min_{P \in \mathbb{U}(\mathbf{a}, \mathbf{b})} \langle P, C \rangle \tag{3}$$

であると定義される．ここで P および C はそれぞれ輸送行列，輸送コストと呼ばれ，ともに $N \times M$ 行列である (ここで N および M は点群 $\mathbf{a}, \mathbf{b}$ に含まれる点の数)．C_{ij} は点 x_i と y_j のユークリッド距離の二乗 $\|x_i - y_j\|^2$ である．P_{ij} は点 x_i の質量 a_i のうち，y_j に「輸送される」質量である．Wasserstein 距離は P と C の内積，すなわち輸送に必要な総コストが最小になるように輸送行列 P を決めることで計算される．

正しい輸送 (過不足のない輸送) であるためには輸送行列は輸送元 $\mathbf{a}$ と輸送先 $\mathbf{b}$ において質量保存則を満たす必要がある．これは P の各 i 行の和が a_i かつ P の各 j 列の和が b_i になる，という制約条件として表現される．すなわち

$$\mathbb{U}(\mathbf{a}, \mathbf{b}) = \{P \in \mathbb{R}_+^{N \times M} \mid (P \mathbb{1}_M = \mathbf{a}) \wedge (P^{\mathrm{T}} \mathbb{1}_N = \mathbf{b})\} \tag{4}$$

この制約条件は線形制約であるので，結局 $W(a, b)$ の計算は線形計画法 (LP) に帰着できる．

この定義では $N \neq M$ の場合でも well-defined であり，さらに点群中の添字のシャッフルを行った場合でも最適化の結果は変わらないため，前節で述べた点群間の距離に対する要求を満たしている．また，W は距離の公理を満たすことも知られている [5].

2.3 問題設定

K 個の点群を含むような点群データベース $\mathcal{D} = \{A_i \mid i = 1, \cdots, K\}$ を考える．一般に，各点群は異なる数の点からなるような点群である．本研究では，問い合わせ点群 B に対して，$\mathcal{D}$ 中，最も B と類似した点群を見つける問題を考える．すなわち，

$$A^* = \mathrm{argmin}_{A_i \in \mathcal{D}} W(A_i, B) \tag{5}$$

なお，以降で述べる方法を top-k 検索などに拡張することは容易である．

3 類似度検索アルゴリズム

上述したように，Wasserstein 距離の計算は LP を解く必要があるが，一般的な LP の解法ではなく，問題の構造を利用した高速アルゴリズムが知られている (この LP の解法については例えば [5] などに詳しい．また Python Optimal Transport library (POT) [6] などのライブラリの実装を用いることができる)．ところが，データベースに含まれる点群の数 K が大きく，また点群に多くの点が含まれている (N や M が大きい) 場合には非常に計算コストが大きくなる．実際，図 1 に示すように，A と B に含まれる点の数の積 MN に対して，Wasserstein 距離の計算時間は急激に増大する．

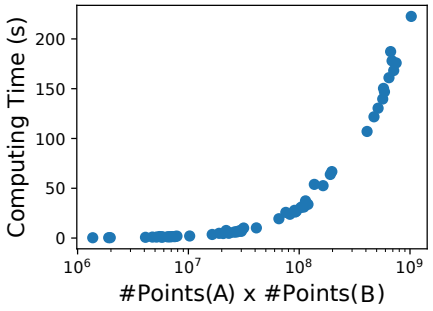

図 1 Wassersten 距離 $W(A, B)$ の計算時間．横軸は A と B に含まれる点の数の積 (Python OT ライブラリ，CPU 計算)

後述する前処理済みの KITTI データセットのすべての点群に対して Wasserstein 距離を計算したところ，16 時間という膨大な時間を要した．

3.1 Sinkhorn アルゴリズム

Wasserstein 距離を近似するアルゴリズムとして Sinkhorn アルゴリズム [7] が知られている．これは点群 A と B の重み $\mathbf{a}$，$\mathbf{b}$ およびその輸送コスト行列 C を入力としてアルゴリズム 1 のように計算される．ここで $\varepsilon > 0$ はアルゴリズムのパラメータである．ε は大きいほど数値的安定性が向上する一方で，近似精度が低下する．近似誤差は ε に比例する [5]．収束条件については後述する．

Algorithm 1: Sinkhorn アルゴリズム．計算はすべて要素ごとに行う．

Data: $\mathbf{a}$, $\mathbf{b}$, C, $\varepsilon > 0$
Result: Wasserstein 距離の近似値

$\mathbf{v} \leftarrow \mathbb{1}_N \in \mathbb{R}^N$, $K \leftarrow \exp(-C/\varepsilon) \in \mathbb{R}^{n \times N}$
while *Not converged* **do**
  $\mathbf{u} \leftarrow \mathbf{a}/K\mathbf{v}$ // 要素ごとの除算
  $\mathbf{v} \leftarrow \mathbf{b}/K^T\mathbf{u}$
return $\langle \mathbf{a}, \log \mathbf{u} \rangle + \langle \mathbf{b}, \log \mathbf{v} \rangle$

a) 理論的背景・収束条件

Sinkhorn アルゴリズムは Wasserstein 距離計算のための LP の双対問題

$$\max_{\mathbf{f}, \mathbf{g}} \langle \mathbf{a}, \mathbf{f} \rangle + \langle \mathbf{b}, \mathbf{g} \rangle \qquad \mathbf{f} \in \mathbb{R}^N, \mathbf{g} \in \mathbb{R}^M \tag{6}$$

$$\text{s.t.} \quad f_i + g_j \leq C_{ij} \qquad \forall i, j \tag{7}$$

の制約条件を緩めて滑らかにした制約なし凸最適化問題

$$\max_{\mathbf{f}, \mathbf{g}} \langle \mathbf{a}, \mathbf{f} \rangle + \langle \mathbf{b}, \mathbf{g} \rangle - \varepsilon \sum_{i,j} \exp\left(\frac{f_i + g_j - C_{ij}}{\varepsilon} \right) \tag{8}$$

に対する block coordinate ascent である．ここで $\mathbf{f} = \log \mathbf{u}$,

$\mathbf{g} = \log \mathbf{v}$ という関係がある．したがってこの制約なし最適化問題の目的関数を $F(\mathbf{f}, \mathbf{g})$ とした場合，反復における前のステップの解 $\mathbf{u}'$ および $\mathbf{v}'$ に対して，目的関数の変化量 $|F(\log \mathbf{u}, \log \mathbf{v}) - F(\log \mathbf{u}', \log \mathbf{v}')|$ が十分小さくなった場合に反復を打ち切ればよい．

3.2 検索アルゴリズム

本研究では点群データベース $\mathcal{D}$ に対して以下のような類似検索アルゴリズムを実装した．

- $\mathcal{D}$ 中の各点群 A_i に対して Sinkhorn アルゴリズムで問い合わせ点群 B との Wasserstein 距離の近似値を計算する．
- 計算された距離がこれまででもっとも良かった場合，それを現在の検索結果とし，そのときの距離を W_{best} とする．
- すべての点群の中で距離が最も小さかったものを検索結果とする．

a） 高 速 化

Sinkhorn アルゴリズムは行列・ベクトル演算の反復アルゴリズムであるため，GPU を用いて高速に演算することができる．また，Sinkhorn アルゴリズムの実行中，$F(\log \mathbf{u}, \log \mathbf{v}) + \varepsilon \geqq W_{\text{best}}$ となった段階で計算を途中で安全に打ち切ることができる．

4 Wasserstein 距離の妥当性

4.1 データセット

評価には KITTI データセット [4] を用いる．このデータセットには LiDAR も用いて道路環境をセンシングした点群データが 43222 シーン含まれている．各点群に含まれる点の数は平均でおよそ 12 万点である．そのような点群同士の Wasserstein 距離を計算するためにはコスト行列 C を計算する必要があるが，この行列のサイズはおよそ 12 万 $\times$ 12 万 であり，(GPU 上，もしくはメモリ上に) 確保することは不可能である．したがって，データ点数を何らかの方法で削減する必要がある．本稿では (1) リサンプリングによる削減 (2) 道路環境を考慮した削減，を行った．これらについて以下で説明する．

リサンプリングによる削減は文字通り，ランダムサンプリングを行うことで点を削減する方法である．道路環境を考慮した削減では，自動車などの道路上のオブジェクトの配置が類似したシーンを検索する，というアプリケーションを想定し，以下のような削減を行った．まず，地面及び高所に対応する点群を消去した．これは KITTI データの場合では z 座標が $-1.3 \leqq z \leqq 1$ となる点以外を除去することで得た．さらにセンサからの距離が遠い点を除去した．今回はセンサ座標 (原点) からの距離が 20 以上の点を除去した．

4.2 結果 1: リサンプリングの影響

前節では点の数を削減するための方策として，リサンプリングを挙げた．リサンプリングによって現実的なリソースで Wasserstein 距離を計算できるようになる (図 1) が，一方で本来計算したかった Wasserstein 距離との誤差も大きくなると予想される．そこで,「道路環境を考慮した削減」を行った

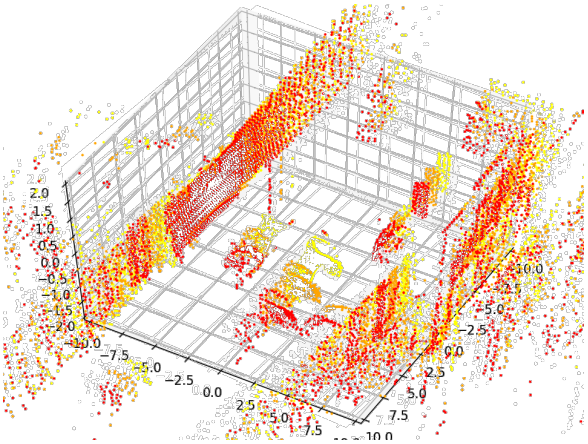

図 2 KITTI データに含まれる点群データの一例 (見やすさのため，$z \leqq -1.3$ の点は除去している．また，時間的に連続した 3 フレームを異なる色で示した)

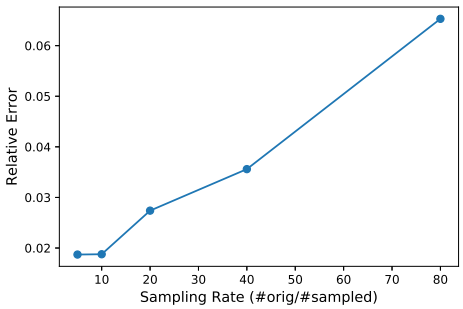

図 3 リサンプリングを行った際の Wasserstein 距離の相対誤差

KITTI データセットに対して，リサンプリングの割合を様々に変化させて，本来の Wasserstein 距離 W_{true} からの相対誤差 $|W(A', B') - W_{\text{true}}|/W_{\text{true}}$ を測定し，図 3 に結果を示した (A', B' はリサンプリングされた点群)．この結果より，5–10 点に 1 点の割合でリサンプリングした場合は相対誤差は 2%未満であるが，それを 80 点に 1 点まで削減すると相対誤差はおよそ 6.5%まで増加した．以降では計算時間と精度の割合を考慮して 20 点に 1 点の割合でリサンプリングを行う．

4.3 結果 2: 検索結果の検証

次に，KITTI データセットの中からクエリとなるシーンを (ランダムに) 選択し，KITTI データセット全体の点群の中で最もクエリ点群との類似したシーンの検索を行った．なお，KITTI データセットには 155 本の軌跡が格納されており，それぞれが各時刻での点群データを格納しているが，クエリとなるシーンを含む軌跡については検索対象から除外した (これは検索結果が同一の軌跡の同時刻の点群になってしまうことを防ぐため)．

図 4 にクエリ点群，およびクエリとの Wasserstein 距離が最小となるデータベース中の点群を示した．最も距離の近い点群とクエリの間にはいくつかの類似点が見られる．

- 両シーンにおいて，左手方向 ($x = -5$ 付近) に車両と思われるオブジェクトが見える．ただしクエリでは 1 台，検索結果では 2 台の車がある．

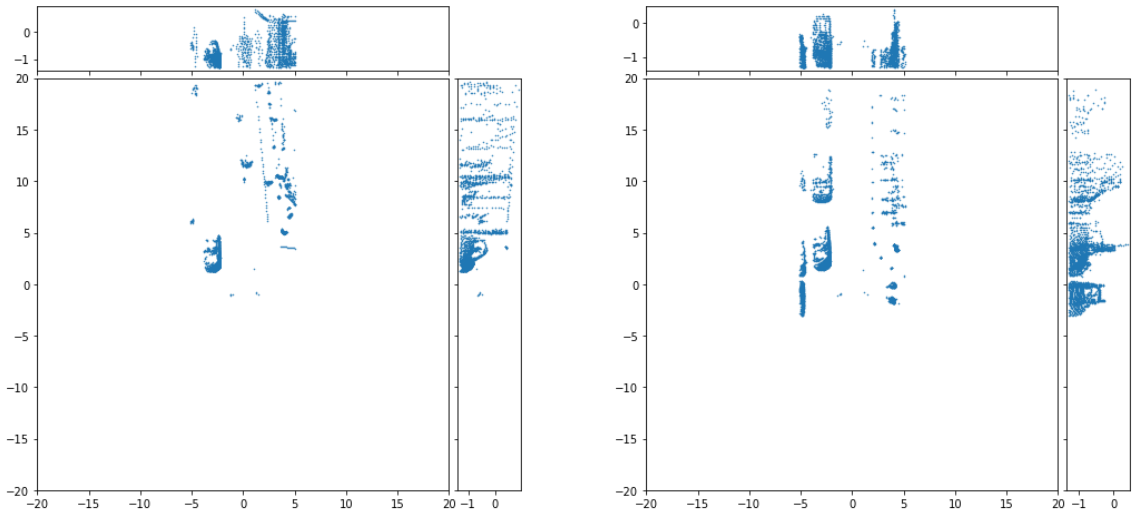

図 4　検索結果 (左: クエリ点群，右: KITTI データセット中のクエリ点群との Wasserstein 距離が最小になる点群)

- 両シーンにおいて，右手方向 ($x = 5$ 付近) に類似の影が見える．これは街路樹などである．

このように Wasserstein 距離を用いることで類似シーンを発見することができる，ということが確認できた．一方で，自動運転開発などのアプリを考えた場合，街路樹などまでマッチさせる必要はないだろう，という議論もあり得る (道路上のオブジェクトの位置関係が類似していればよい，という考え方)．適切な前処理や距離指標の検討は今後の重要な課題である．

4.4　結果 3: 計算速度に関する検証

図 4 の検索には 16 時間という長い時間を要した (この計算には Sinkhorn アルゴリズムではなく，先述した Python OT ライブラリの `ot.emd` 関数を用いた．これは [8] に基づくアルゴリズムである)．これを Sinkhorn アルゴリズム (GPU 併用) を用いた検索アルゴリズムに置き換えたところ，8 分程度まで計算時間が短縮された．したがって，Wasserstein 距離による類似検索は現実的な時間で可能である．

4.5　議　　論

一般的に，LiDAR データは極めて多くの点を含むため，Wasserstein 距離を定義どおり計算することは現実的ではない．しかしながら，リサンプリングや Sinkhorn アルゴリズムを組み合わせることによって，ある程度の精度を保ちながら現実的な処理時間で検索結果を得ることができることが示唆された．ただし，検索をミリ秒〜数秒オーダーまで改善するためには何らかの索引構造の開発が不可欠であろう．また，実際の検索結果は図 4 で示したように，定性的にもある程度，クエリ点群との類似性を持っている．一方で，応用に合わせて距離指標を変更する，もしくは前処理を改良する，などの工夫を行う必要が出てくることも想定される．道路環境の類似度の適切な評価指標および前処理の開発は今後の課題である．

5　関 連 研 究

Wasserstein 距離は最適輸送理論 (Optimal Transport) と深い関連を持つ．最適輸送理論は OR 分野で長い歴史を持つが，これらの最近の進展を含めた文献として [5] がある．Wasserstein 距離を近似する代表的なアルゴリズムとして，本研究でも用いる Sinkhorn アルゴリズムがある．これは [7] で再評価された matrix scaling アルゴリズムであり，反復計算で Wasserstein 距離を近似する．

CNN などの深層学習技術によって画像に対する Wasserstein 距離を近似しよう，という試みもある [9]．ニューラルネットワークによる近似技術は，柔軟性が高い一方，近似精度の保証がないという欠点がある．[10] では Wasserstein 距離に対する索引を構築することを試みている．この研究では Quadtree ベースの索引を構築し，Sinkhorn アルゴリズムで線形探索したケースよりも高速である，と結論しているが，比較されている Sinkhorn アルゴリズムは GPU 実装されておらず，評価が妥当であるとは言えない．

点群を格納するためのデータベースシステムとしては，PostgreSQL の pointcloud 拡張がある [11]．このようなシステムは今回は用いていないが，現実的なシステム構築の際には DBMS の上に類似度検索を実装する必要があるだろう．

近年，自然言語処理の分野では Word Mover's Distance (WMD) [12] が用いられることも多いが，これは文書を word2vec などの単語埋め込みベクトルの点群とみなし，Wasserstein 距離を用いるものである．

6　ま　と　め

本研究では LiDAR などから得られる点群データに対する，類似度検索の検討を行った．類似度関数として Wasserstein 距離も用い，Sinkhorn アルゴリズムをもちいて近似距離計算を

行った．LiDAR データ (KITTI データ) に対して 1-NN 検索を行い，定性的な評価を行った結果，以下の知見が得られた: Wasserstein 距離は可変長の点群を比較可能である，という利点があるものの，実際の道路環境の点群の検索に用いる際には，点群を適切かつ慎重に前処理する必要がある．本研究では背景部分を一定のルールに従って削除することで，道路上のオブジェクトのみを取り出す，という前処理を行った．どのような前処理が妥当か，という問題はアプリケーションにも依存する問題であり，この部分は今後の大きな課題である．

また今回は索引を構築せずに GPU による高速計算で検索を行った．関連研究の項でも述べたとおり，Wasserstein 距離に対する索引化技術も (ごく最近) 提案されており，これらとの比較を行うことも重要な課題である．

文　献

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, arXiv:1612.00593, 2016.

[2] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep Hierachical Feature Learning on Point Sets in a Metric Space, arXiv:1706.02413, 2017.

[3] 井川元, 渡辺陽介, 高田広章. 高精度地図データおよび点群データの検索効率化手法. In DEIM'19, 2019.

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. International Journal of Robotics Research (IJRR), 2013.

[5] G. Peyrè and M. Cuturi. *Computational Optimal Transport*. No. ArXiv:1803.00567. 2018.

[6] R. Flamary and N. Courty. POT: Python Optimal Transport library, 2017.

[7] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 26, pp. 2292–2300. Curran Associates, Inc., 2013.

[8] N. Bonneel, M. v. d. Panne, S. Paris, and W. Heidrich. Displacement interpolation using lagrangian mass transport. ACM Trans. Graph., Vol. 30, No. 6, pp. 158:1–158:12, December 2011.

[9] N. Courty, R. Flamary, and M. Ducoffe. Learning wasserstein embeddings. In International Conference on Learning Representations, 2018.

[10] Y. Dong, P. Indyk, I. Razenshteyn, and T. Wagner. Scalable nearest neighbor search for optimal transport, 2019.

[11] Postgresql pointcloud extension. `https://github.com/pgpointcloud/pointcloud`.

[12] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, pp. 957–966. JMLR.org, 2015.