David Yenicelik
ETH Zürich

Eidgenössische Technische Hochs
Swiss Federal Institute of Technol

A dissertation submitted to ETH Zürich
in partial fulfilment of the requirements for the degree of
Master of Science ETH in Computer Science

ETH Zürich
Data Analytics Lab
Institute of Machine Learning
Zurich, 8006
Switzerland

Email: yedavid@ethz.ch

November 17, 2019

Declaration

I David Yenicelik of ETH Zürich, being a candidate for the M.Sc. in Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 14,235

Signed:

Date:

Abstract

Write a summary of the whole thing. Make sure it fits in one page.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In Natural Language Processing (NLP), bilingual lexical induction (BLI) is a problem of inferring word-to-word mapping between two languages. While supervised BLI may be learned trivially from a dictionary, unsupervised BLI is highly non-trivial, and serves as a backbone to many unsupervised Neural Machine Translation (NMT) systems, without which the overall MT performance drastically drops [**?**] [**?**]. In addition to the unsupervised setting, words in a source language A often do not carry a one-to-one correspondence with words in a target language B. This further increases the difficulty of finding a (bijective) mapping between the two embedding spaces.

1.1 Scope of Work

Continuing where [**?**] left off, we want to investigate the performance of using normalising flows to model unsupervised lexicon matching between two probability distributions, which are defined by Gaussian emebddings, which have a one-to-one correspondence to tokens in the respective languages. We aim to use Gaussian embeddings for the robustness, and better integration into the probabilistic perspective. The method discussed in the paper also relies a lot on the (semi-)supervised loss component. We aim to investigate why this is the case, and would like to revise a method which is more robust

in the fully unsupervised setting.

Minimal goals: We propose the following steps on achieving this goal.

1. Replicate the algorithms and models which can generate through Gaussian embedding [**?**]. Do one sanity check by doing a sanity check on one of (SimLex / WordSim)

2. Replicate "Density matching for bilingual word embedding" to setup a baseline normalising flow between vector-word-embeddings [**?**].

3. Define loss between predicted and target embeddings (if change in definition is necessary)

4. Change the embeddings in point 2. to use Gaussian embeddings. Implement Loss functions found in point 3.

Extended goals: If the above points provide good performance , we would like to expand on the below points.

1. Implement a fully unsupervised extension by investigating the shortcomings of [**?**].

2. Investigate "deeper" normalising flows than the linear flow in [**?**] as such [**?**] [**?**].

Contingency Plan / Further extended goals: Implementing the following points would allow for an applied perspective of this approach, showing that this methodology allows for more robust mapping, also outside the field of NLP.

1. Train embeddings for job-systems ESCO and AUGOV using Skip-Gram or Co-Occurence-matrix based. Do a sanity check.

2. Train Gaussian embeddings for job-systems. Do a sanity check.

3. Generate a small validation dataset between the European- and Australian job-system.

4. Setup some baseline algorithms based on NLP, graph-matching, colinear-PCA for matching as a non-NLP benchmark environment. Compare against above-proposed methods.

5. Find a normlising flow model to transform one job system into another.

Chapter 2

Background

We provide a short introduction to the background work.

2.1 Gaussian Embeddings

The problem that Gaussian Embeddings try to resolve is to find a suitable embedding space for entities, such as word-tokens.

In the context of NLP, the general goal is to map every word type w that is included in some dictionary $\mathcal{D}$ in the latent space, such that w is close to all the context words c from another dictionary $\mathcal{C}$. This follws the (CITE THE RANDOM GUY) ASSUMPTION, which says that words in similar contexts have similar neighborhoods.

In an unsupervised context, we observe a sequence of words $t(w)_i$ for each word-token w_i and their contexts $c(w)_i$.

We want to use some rank-based Energy, not absolute energy, such that we can make sure that neighboring words are pushed together, and two "random" words are pushed further away from each other.

The above two problems can be regarded as finding an invertible transformation from one (embedding) space $\mathcal{X} \in \mathbf{R}^d$ into another $\hat{\mathcal{X}} \in \mathbf{R}^d$. Normalising flows [**?**] [**?**] have proven to be a powerful tool in modeling such relations. As

such, the aim of this project is to find a model based on normalising flows which is able to find an an invertible mapping f from $\mathcal{X}$ to $\hat{\mathcal{X}}$. To keep the discussion focused, this thesis deals with the problem of finding a model for bilingual lexicon matching.

Word representations via Gaussian Embeddings

Gaussian Embeddings have been first proposed in the context of words, although prior work has been adapted in embedding matrix-rows into a mean and standard deviation (CAN CITE THE PEOPLE IN THE PAPER 2007, 2008) . It is a continuous probabilistic relaxation of the otherwise common discrete point vectors. Each word is represented by a Gaussian distribution in high-dimensional space, allowing to better capture uncertainty and a representation and it's relationships. It can also express asymmetric relations more naturally than dot-products or cosine similarity, and enables for better-parametrized rule between decision boundaries.

Fitting Gaussian Mixture Models on embeddings have been done in order to apply Fisher kernels to entire documents.

Because this is an unsupervised learning task, we must use an energy function which is incorporated within a loss function that we try to minimize. The energy function describes (dis-)similarity between two items. The authors propose the following energy functions to derive a Gaussian word-embedding.

$$L_m(w, c_p, c_n) = max(0, m - E_\theta(w, c_p) + E_\theta(w, c_n) \tag{2.1}$$

Here, w is the word we want to sample, c_p is a "positive" context word, i.e. a word that co-occurs with the word w, and c_n is a "negative" context word, i.e. a word that does not co-occur with the word w. Usually the negative context words is sampled randomly from the corpus. The loss function reminds of a hinge-loss in logistic regression.

The authors propose two possible ways to learn the mean and variance of the Gaussian embeddings. They argue that the empirical covariance is not the

most effective method of deriving the words as Gaussian embeddings. This does not allow for inclusion between ellipsoids

Symmetric similarity: expected likelihood or probability product kernel We can use any kernel (which is symmetric by definition) to derive at an energy function. For two Gaussians $f(x)$, $g(x)$, the inner product is defined as:

$$E_\theta(w, c) = \int_{x \in \mathcal{R}^d} f(x)g(x)dx \tag{2.2}$$

$$= \int_{x \in \mathcal{R}^d} \mathcal{N}(x; \mu_w, \Sigma_w)\mathcal{N}(x; \mu_c, \Sigma_c)dx \tag{2.3}$$

$$= \mathcal{N}(0; \mu_w - \mu_c, \Sigma_w + \Sigma_c) \tag{2.4}$$

For numerical feasibility and easy of differentiation, we usually maximize the $\log E_\theta(w, c)$ for a given dataset with $w \in \mathcal{W}, c \in \mathcal{C}$. We will not go further in what the specific gradient of this log-energy is.

Asymmetric divergence: KL-Divergence We can use more directional supervision to exploit directional supervision, such as a knowledge graph.

Following energy-function is optimized:

$$-E(w_i, c_j) = D_{KL}(c_j||w_i) \tag{2.5}$$

$$= \int_{x \in \mathcal{R}^d} \mathcal{N}(x; \mu_{w_i}, \Sigma_{w_i}) \log \frac{\mathcal{N}(x; \mu_{c_j}, \Sigma_{c_j})}{\mathcal{N}(x; \mu_{w_i}, \Sigma_{w_i})} dx \tag{2.6}$$

$$= \frac{1}{2}\left(\mathrm{tr}\left(\Sigma_i^{-1}\Sigma_j\right) + (\mu_i - \mu_j)^\top \Sigma_i^{-1}(\mu_i - \mu_j) - d - \log \frac{\det(\Sigma_j)}{\det(\Sigma_i)}\right) \tag{2.7}$$

Because of the loss function, this can entail information such as "y entails x" as a soft form of inclusion between two datasets (if KL divergence is used). If

a symmetric loss function is used, then this would most likely lead to overlap (IS THIS TRUE...???)

Uncertainty calculation: In contrast to the empirical standard deviation as an uncertainty measure, we can now calculate the uncertainty of the inner product (i.e. the distribution $P(z = x^T y)$ using the following formula

$$\mu_z = \mu_x^T \mu_y \Sigma_z = \mu_x^T \Sigma_x \mu_x + \mu_y^T \Sigma_y \mu_y + \mathrm{tr}\left(\Sigma_x \Sigma_y\right) \tag{2.8}$$

We then get an uncertainty bound, where c denotes the number of standard deviations away from the mean.

$$\mu_x^\top \mu_y \pm c\sqrt{\mu_x^\top \Sigma_x \mu_x + \mu_y^\top \Sigma_y \mu_y + \mathrm{tr}\left(\Sigma_x \Sigma_y\right)} \tag{2.9}$$

We can learn the parameters Σ and μ for each of these embeddings using a simple gradient-based approach, where we set hard constraints on

$$\|\mu_i\|_2 \leq C, \forall i \tag{2.10}$$

$$mI < \Sigma_i < MI \tag{2.11}$$

The method shows competitive scores to the Skip-Gram model, although usually only with minor improvements depending on the benchmark-dataset.

2.2 General random information

For implicit generative models, we can measure the performance in two ways.

Assuming we have two probability density distributions, the true data-distribution $p*$ and the approximate posterior q_θ whose goal is to mimick the original data distribution $p*$.

One way is to apply the probability difference.

$$r_\phi = p * -q_\theta \tag{2.12}$$

This in general, however is not very intuitive, and does not easily allow to differentiate between mutliple differences. Loss functions that make use of this logic in clude the Max-Mean discrepancy, or moment-matching loss functions.

Another way it the probability ratio. This ratio is much more intuitive in that one can compare the performance of how different approximate posteriors match the true underlying data distributiion.

$$r_\phi = \frac{p*}{q_\theta} \tag{2.13}$$

Examples include the f-diverence, class probability estimate and the Bregman divergence.

—

What do GANs usually do. We assume a simple vector z sampled from a feasibly-inferable distribution z.

$$z \; p(z) \tag{2.14}$$

$$x^{gan} = f_\phi(z) \tag{2.15}$$

In general, we try to minimze an energy function, using some sort of reward R and action a taken in some state s. [THIS IS COMPLETE BULLSHIT]

$$\mathbf{E}_{\phi(a|s)}\left[R(s,a)\right] - KL\left[\pi_\theta(a|s)||p(a)\right] \tag{2.16}$$

2.3 Normalising Flows

In machine learning, we usually distinguish between fully observed models and latent variable models, where we have exclusively observed variables or both observed and unobserved variables.

In general, a generative model can be described as

$$p(x, z, \theta) = p(\theta) \sum_{i=1}^{N} p(x_i|z_i, \theta)\pi(z_i) \tag{2.17}$$

And then the model is supplemented by a learning principle such as gradient-based learning (backpropagation), or structural learning (model selection using Monte-Carlo). In general, we want to estimate a probability density distribution by using a stochastic gradient estimator.

$$\nabla_\phi E_{q_\phi(z)}\left[f_\theta(z)\right] = \nabla \int q_\phi(z) f_\theta(z) dz \tag{2.18}$$

$$= \mathbf{E}\left[\nabla f_\theta(g(\epsilon, \phi)\right] \tag{2.19}$$

$$q(z') = q(z) \left| \det \frac{\delta f}{\delta z} \right|^{-1} \tag{2.20}$$

We can now stack multiple bijective and invertible functions together to go from a complex distribution to a simple one (and vice-versa).

(CITE REZENDE ET AL TUTORIAL ON DEEP GENERATIVE MODELS)

$$z_K = f_K \circ \ldots \circ f_2 \circ f_1(z_0) \tag{2.21}$$

where the mass-preserving transformation of each function is

$$\log q_K(z_k) = \log q_0(z_0) - \sum_{k=1}^{K} \log \det \left| \frac{\delta f_k}{\delta z_k} \right| \tag{2.22}$$

The loss function which we try to maximize as a product of this then becomes (through the use of expected stochastic gradients.

$$\mathcal{L} = \mathbf{E}_{q_0}(z_0) \left[\log p(x, z_K) \right] - \mathbf{E}_{q_0}(z_0) \left[\log p(x, z_K) \right] \tag{2.23}$$

Where we used the reparametrization trick in the second equality. We can then use gradient-estimates to miminize the respective log-likelihood.

The REINFORCE (reparametrization trick) has been used in reinforcement learning

$$\nabla_\theta \mathbf{E}_{x \approx p_\theta(x)} \left[f(x) \right] \nabla_\theta p_\theta(x) = p_\theta(x) \nabla \nabla_\theta \log p_\theta(x) \tag{2.24}$$

Rich families of posteriors can be formed.

Generally, if we have N variables, then 3 possible posteriors are (1) the fully connected graph which describes correlations between all variables, (2) a structured approximation model and (3) a fully factored model.

The posterior probabilities of these items respectively are:

1. $$q^*(z|x) \propto p(x|z)p/z)$$

2. $$q(z) = \prod_k q_k(z_k | z_{j\, j \neq k}$$

3.

$$q(z|x) = \prod_k q(z_k)$$

(Rezende 2015) A simple distribution is transformed into a more complex one by applying a sequence of invertible transformations until a desired level of complexity is attained.

In the above equation, the KL-divergence is the loss between the approximate posterior and the prior distribution (which acts as a regualizer). The second term, which is the expected log-likelihood is the reconstruction error.

Inference with normalizing flows provides a tighter, modified variational lower bound with additional terms that only add terms with linear time complexity.

in the asymptotic regime, this is able to recover the true posterioir distribution

Approximate posterior distribution of the latent variables $q_\phi(z|x)$

$$\begin{aligned}
\log p_\theta(x) &= \log \int p_\theta(x|z)p(z)dz && (2.25)\\
&= \log \int \frac{q_\theta(z|x)}{q_\theta(z|x)} p_\theta(x|z)p(z)dz && (2.26)\\
&= \log \int \frac{p(z)}{q_\theta(z|x)} p_\theta(x|z)q_\theta(z|x)dz && (2.27)\\
&= \mathbb{D}_{KL}\left[q_\phi(z|x)||p(z)\right] + \mathbb{E}_q\left[\log p_\theta(x|z)\right] && (2.28)\\
&\geq -\mathcal{F}(x) && (2.29)
\end{aligned}$$

TODO: WHERE DOES THE LAST $q_\theta(z|x)$ disappear to? where $-\mathbf{F}(x)$ is a free energy function, and where we used Jensen's inequality to obtain the final equation (through $logE_q\left[\frac{p(x,z)}{q(z)}\right] \geq logE_q\left[\log\frac{p(x,z)}{q(z)}\right]$). $p_\theta(x|z)$ is a likelihood function and $p(z)$ is a prior over latent variables, and $q_\theta(z|x)$ is a

(simple) model distribution with which we want to approximate $p(x|z)$. item We try to minimize this loss

To train the model, we need to efficiently (1) compute the derivatives of the expected log-likelihood $\nabla\phi\mathbb{E}_{q_\phi(z)}\left[p_\theta(x|z)\right]$ and (2) choose the richest, computationally-feasible approximate posterior distribution $q(\dot{)}$.

Using Monte Carlo gradient estimation and inference networks, which (when used together), they call *amortized variational inference*. TODO: TIMO SAID THAT THIS IS WRONGLY APPROXIMATING THE TRUE DISTRIBUTION? WHAT ABOUT PSI?

For stochastic backpropagation, we make use of two techniques:

- Reparameterization: The latent variable is reparametrized in terms of a known base distribution and a differentiable transofmration. As an example, if $q_\phi(z)$ is a Gaussian distribution $\mathbf{N}(z|\mu,\sigma^2)$ with trainable parameters $\phi = \mu, \sigma^2$, then we can reparametrize the variable z as

$$z \sim \mathbf{N}(z|\mu,\sigma^2) \iff z = \sigma + \sigma\epsilon, \epsilon \sim \mathbf{N}(0,1) \tag{2.30}$$

- Backpropagation with Monte Carlo. We backpropagated w.r.t. the parameters ϕ of the variational distribution using a Monte Carlo apprxoimation. This is our samples batch.

$$\nabla\phi\mathbb{E}_{q_\phi(z)}\left[f_\theta(z)\right] \iff \nabla\phi\mathbb{E}_{\mathbf{N}(\epsilon|0,1)}\left[\nabla_\phi f_\theta(\mu+\sigma\epsilon)\right] \tag{2.31}$$

  where we have simply rewritten z using the reparametrization trick.

For continuous latent variables, it has the lowest variance among competing estimators [CITEEE].

Normalizing flows are most often used in the context of inference network.

An inference network learns an inverse map from observations to latent variables. The simplest latent variable is a Gaussian distribution.

Rezende 2015 propose a *Deep Latent Gaussian Model*, which consists of a hierarchy of L layers of Gaussian latent variables z_l for layer l.

$$p(x, z_1, \ldots, z_L) = p(x|f_0(z_1)) \prod_{l=1}^{L} p(z_l|f_l(z_{l+1})) \tag{2.32}$$

We induce priors over each latent variable $p(z_l) = \mathbf{N}(0, I)$, and the observatoin distribution $p_\theta(x|z)$ is any distribution that is conditioned on z_1 and is parametrized by a neural network.

This model is very general, and captures includes other models, such as factor analysis and PCA; non-linear factor analyiss, and non-linear Gaussian belif networks as special cases (Rezende 2014, cited in Rezende 2015)

We know that $\mathbf{D}_{KL}\left[q||p\right] = 0$ is achived when $q_\phi(z|x) = p_\theta(z|x)$ (i.e. the approximate q matches the true posterior distribution). IS THIS THE ONLY TIME THIS CAN BE ACHIEVED?

A normalising flow captures more flexibel distributions.

Summary normalising flows:

A normalizing flow is a set of basic rules for transofmration of densiities, considers an invertible, smooth mapping

$$f : \mathbf{R}^d \to \mathbf{R}^d$$

where the input dimensioinality d_0 and output dimensionalities d_L match. The inverse is denoted $f^{-1} = g$, i.e. the composition $g \circ f(z) = z$. When

we want to transform a random variable $z' = f(z)$, we receive the following distribution:

This stems from the Identity that the probability mass must be conversed amongst operations:

$$p_x(x) = \frac{p_z(z)V_Z}{V_X} \tag{2.33}$$

$$z' = f(z) \tag{2.34}$$

$$\frac{z'}{q(z')} = \frac{f(z)}{q(z)} \tag{2.35}$$

WHERE DOES THE PRIME COME FROM

We can then construct arbitrarily complex densities by composing several simple maps and successively applying the above mass-preserving operation.

$$z_K = f_K \circ \ldots \circ f_2 \circ f_1(z_0) \tag{2.36}$$

We then apply the log-function to have tractable probability functions.

Probability mass must be conserved. From the change of variable formula, we know that taking infinitesimal c:hanges toward the direction of the final probabiliity distribution, we get

$$\int z'q(z')dz' = \int f(z)q(z)dz \tag{2.37}$$

$$q(z') = q(z)\ln \left| det \frac{\delta f(z)}{\delta z'} \right| \tag{2.38}$$

There are different ways to use normalising flows:

Although computing the above gradient is expensive ($O(n^3)$ complexity), we can use the Matrix determinant lemma, and construct out mapping matrices in a special way as follows (https://blog.evjang.com/2018/01/nf1.htm)

$$det(A) = det(L + V \cdot D \cdot V^T) \tag{2.39}$$
$$= det(L)(1 + (VD^{\frac{1}{2}})^T(L^{-1}D^{\frac{1}{2}}V)) \tag{2.40}$$

$$W = PL(U + diag(s)) \tag{2.41}$$

Then the cost of computing $det(W)$ becomes $O(c)$ instead of $O(c^3)$. (CITING GLOW Glow : Generative Flow with Invertible 1 x 1 Convolutions)

where we have used the Matrix Determiinant Lemma for the second equality, with L is a lower triangular matrix, and D is a diagonal matrix.

subsectionExtensions on Normalising Flows

(https://www.shakirm.com/slides/DeepGenModelsTutorial.pdf)

Ever since Normalising flows were proposed, a number of modified versions were presented by [CITE].

Each one calculates the consecutive flow based on a different composition of the previous latent variables z_k.

Planar Flow

The planar flow is a simple sequential flow with invertible piecewise linear transformations.

$$z_k = z_{k-1} + uh(w^T z_{k-1} + b) \tag{2.42}$$

Real NVP (2017)

The input for each step is split into two parts. The intuition behind this is similar to a residual network, which includes logic that allows the gradient to flow back easier.

$$y_{1:d} = z_{k-1,1:d} \tag{2.43}$$

$$y_{d+1:D} = t(z_{k-1,1:d}) + z_{d+1:D} \odot \exp(s(z_{k-1,1:d})) \tag{2.44}$$

Here, t and s are arbitrary functions, such as deep neural networks or simple linear transforms.

Provides a set of stably invertible and learnable transformations.

This results in an exact log-likelihood calculation.

In the paper, they for the networks s and t, they use residual network with batchnorm and weight normalization.

Inverse AR Flow (2017)

According to the authors, this method scales well to high dimensional latent spaces. Each transformation inside the flow is based on an autoregressive neural network.

At each timestep, all prior variables are taken into consideration.

$$z_k = \frac{z_{k-1} - \mu_k(z_{`k}, x)}{\sigma_k(z_{`k}, x)} \tag{2.45}$$

Non-Linear Independent Components Estimation (NICE)

Another volume-preserving flow is the NICE flow. First, we arbitrarily partition the latent vector into two components $z = (z_A, z_B)$.

$$f(z) = (z_A, z_B + h_\lambda(z_A)) \tag{2.46}$$
$$g(z') = (z'_A, z'_B + h_\lambda(z'_A)) \tag{2.47}$$

Here, h_λ can be chosen in such a way that h is a deep neural network with parameters λ

LOL, COULD WE JUST CREATE A NEW ONE WITH MORE PROPERTIES??

Multiple

Normalising flows [**?**], [**?**] are a statistical technique where a series of invertible transformations f_t are applied to a simple distribution $z_0 \sim q_0(z)$, to yield increasingly complex distributions $z_t = f_t(z_{t-1})$, s.t. the last iterate z_T has the desired and more flexible distribution. As long as we can efficiently compute the Jacobian determinant of the transformation bijection f_t, we can both (1) evaluate the density of our data (by applying an inverse transformation and computing the density in the base distribution), and (2) sample from our complex distribution (by sampling from the base distribution and applying the forward transformation) These can be used for classification and clustering [**?**], [variational inference tasks [**?**] such as image-generation [**?**]], enriching the posterior (and prior!) [**?**], and density estimation [**?**].

Glow: Generative Flows with Invertible 1x1 Convolutions (2018)

We use 1x1 convolutional layers as a generalization of the permutation in sequential flows with lower triangular learnable rotation matrices (assuming that the number of input and output channels are equivalent).

TODO: Convolution operators and reduction to permutation operators

$$\log \left| \det \left(\frac{d\text{conv2D}(h; W)}{dh} \right) \right| = h \cdot w \cdot \log|\det(W)| \tag{2.48}$$

(read the paper a bit more before citing this, but cite this: https://arxiv.org/pdf/1901.08624.pdf)

The authors cite strong qualitative experimental results in generating faces, thus empirically proving that higher dimensional flows (R^d) are effective and learn a continuous latent space.

Flow++ (2019)

The Flow++ model describes three problems with existing methods.

1. uniform noise is a suboptimal dequantization which hurts training loss and generatlization (i.e. training points are mapped to discrete points in space which have the full probability mass)
2. the current models are not expressive enough
3. convolutional layers in coupling are not powerful enough

The authors respectively propose three ways to

1. variational flow-based quantizatioin (WUT?)
2. logistic mixture CDF coupling flows (WUT?)
3. self-attention in conditioning of networks

The coupling layers can be described as

$$y_1 = x_1 \tag{2.49}$$

$$y_2 = x_2 \cdot exp(a_\theta(x_1)) + b_\theta(x_1) \tag{2.50}$$

where the functions a_θ and b_θ are learnable functions (possibly normalizing flows which). These have to be invertible affine transformations

It is the first model which starts to close the gap between autoregressive models and flow-based models.

Chapter 3

Related Work

This work deals with some summary on how unsupervised bilingual lexicon matching was achieved in past papers.

I will give a short summary of papers sorted by year of appearance, and what additional contributions and observation were added since the last iteration.

Some terminiology before we start

online data implies that some sort of parallel corpora is available

offline data implies that we use pre-trained monolingual embeddings to arrive at a toen-translation task

The general goal of bilingual lexicon matching is to learn a shared embedding space where words possessing simiilar meanings are projected to nearby points.

Most existing methods focus on minimizing the distance between the two token-datasets using some (variation) of Wassterstein, Jensen-Shannon divergence, etc.

3.1 Methods using GANs (and other stuff..)

3.2 Methods using Normalising Flows

3.2.1 Density Matching for Bilingual Word Embeddings (Zhou 2019)

3.3 Methods viewing this problem as an Optimal Transport (OT) Problem

3.3.1 Gromov-Wasserstein Alignment of Word Embedding Spaces

Chapter 4

Analysis of the current state of the art

Chapter 5

Our Method

Chapter 6

Evaluation

Chapter 7

Conclusion