Automatic Autoencoding Variational Bayes for Latent Dirichlet Allocation

Zach Colin Wolpe u15094813

STK795 Research Report Proposal

Submitted in partial fulfillment of the degree BCom(Hons) Statistics

Department of Statistics, University of Pretoria

24 February 2019

Abstract

This research report explores the mechanisms behind scalable approximations and parameter estimation for datasets with concealed variables that potentially offer great insight. We leverage modern techniques and machinery for the purpose of achieving extremely efficient estimation. Many topic models require parameter estimation as a consequence of intractability: established methods are while ironed and optimized however extremely computationally expensive. As datasets grow in complexity and dimensionality many of the computational methods fail, thus a dynamic alternative would be serve great benefit. After conducting various experiments the proposed unconventional approach - utilizing modern and topical deep learning methods - delivers inadequate performance: as a corollary of the network over-biasing narrowly specified initial random condition. Under Utopian conditions comparable conclusion are achieved which are generally unattainable. Further, model specification becomes increasingly complex and deeply circumstantially dependant - which is in itself not a deterrent but does warrant consideration.

Contents

List of Figures

Declaration

I, *Zach Wolpe,* declare that this essay, submitted in partial fulfillment of the degree *BCom(Hons) Statistics,* at the University of Pretoria, is my own work and has not been previously submitted at this or any other tertiary institution.

_____

Zach Wolpe

_____

Dr Alta De Waal

_____

September 18, 2019

1 Introduction

Big Data has become so topical, however without sufficient and timely measures to digest and comprehend this data, no additional information is gained. Consider your favourite online blog, news website or social media platform: how often is new material being produced? It's not difficult for even the least tech-savvy individual to see that information is growing at unprecedented rates. How much of this information is structured in such a way as to allow for deriving insight? The user generated nature of online information is inherently chaotic, some document types (like mainstream media outlets) are well-documented, tagged, sorted and curated - yet a batch of a hundred thousand restaurant reviews is slightly less seamless to organize. Vast quantities of unlabelled text data are produced daily at exponentially increasing rates. Unsupervised (and semi-supervised) models are becoming more popular in both academia and industry. Although effective, label text data (such as labeled tweets for sentiment analysis) proves problematic for the following reasons:

- Manually labelling text is cumbersome, expensive and often inaccurate.
- The subjective nature of text labelling omits descriptive information: this leaves the dictation of *what is imperative* at the sole discretion of the labeller; how certain can we be that any individual labeling vast quantities of data could even remain consistent, let along capture the true essence of the text.
- Algorithms trained on one dataset are seldom generalisable to new datasets, as they are highly contextual: the diversity of text online is undeniable - a model that determines tweet sentiments will almost certainly perform poorly if applied to news articles.

Natural language text is unstructured and noisy. The underlying information that is essential in describing text data is often undocumented: what are the key ideas in the document; how does it relate to other texts and whats the true message being delivered. If understood, these hidden (or latent) variables could vastly improve the categorisation and organization of the data. Apart from its unstructured nature, the mathematical properties of text/language is not explicit. Subsequently, performing text-based statistical inference is challenging. How then, do we best mathematically describe text documents through parameterisation and/or function approximation? The incorporation of Bayesian modeling naturally fits this setting, as we'd like to take advantage of prior assumptions of the data's structure. Methods designed to infer latent variables from the data exist. One such class of methods is generative models: in which assumptions are made about how the data was generated in order to find the true distribution of the data [6]. A mathematical dilemma arises when solving for the posterior distribution of the latent variables. Integer intractability calls for approximation techniques to best estimate certain model parameters. Iterative procedures offer a viable solution (wherein iterating until convergence serves as a

valid approximation, including extremely popular sampling techniques like Gibbs sampling) [6]. However these methods fail when working through sizable datasets; simply because they are too computationally expensive.

Variational methods offer an alternative to expensive sampling procedures [6]. It may be possible to turn the simulation problem into one of optimization and thereby potentially improving the speed dramatically. A proposed solution for variational approximation of these intractable integers is using Auto-Encoding Variational Bayes (AEVB), an algorithm to estimate continuous latent variables. This estimation technique leverages an Auto-Encoder (AE) neutral network for pattern recognition, to facilitate a Variational Bayes (VB) approach: optimisation of an approximation to the intractable posterior [9].

An Artificial Neural Network (ANN) learns the relationship (function) between input data x and output data y - particularly learning non-linear relationships. An ANN consists of: input nodes (corresponding to $\mathbf{x}$ independent variables); hidden layers consisting of nodes (which learn weights and biases to approximate relationships between variables); and output layer, where each node is a response variable y [6]. An autoencoder is simply an ANN that learns the relationship between x and x, allowing us to study the relational approximation between x and itself. initially, this appear inherently uninteresting in the conventional sense of observing the output data (since the output is simple a replica of the input). However if we design the ANN such that the number of nodes in the hidden layers are far less than that of the input/output layer, we indirectly force the network to compress the data into a lower dimensional space [6]. If the output accurately resembles the input, we have successfully implemented a dimensionality reduction technique on the data - where the hidden layer/s is/are a less dense version of the same information. By then observing the mapping of the compressed hidden layer to the reconstructed input data we can infer the data generated process [6]. If we instead learn a probabilistic, stochastic model of the input data so that the hidden layer consists of probabilistic distributions as apposed to simple numerical values, we arrive at a *variational* auto-encoder [18].

This variational inference methodology is preferred as it does not rely on full iterative processes to update parameter estimates, but instead updates estimates per a set batch or ever per single datapoint - making for a scalable approximation technique. We can also uncover estimations for the generative process itself, aiding in a series of applications.

For demonstrative purposes we turn to Latent Dirichlet Allocation (LDA) as it perfectly describes an unsupervised, probabilistic, text modelling technique that fits the above-mentioned specifications. Dimensionality reduction is pivotal in deciphering semantic meaning in large texts. LDA is a topic modelling algorithm that reduces dimensionality by extracting latent variables that serve as the topics in the model, subsequently clustering the texts according to similarity. Further, it assumes a generative process [6].

This experimental research is conducted in the attempt to address the subsequent research objectives:

1. *Does the autoencoder LDA provide comparable results - and predictive capability - to the well establish variational inference - online learning - LDA?*
2. *Is it possible to scale the autoencoder to large datasets?*
3. *Does the autoencoder offer significant processing-efficiency advantages as datasets scale?*

2 Auto-Encoding Variational Bayes

Statistical inference becomes increasingly challenging as our models grow in complexity. To understand a variational autoencoder, first we must distill its constituents. A graphical model is a way to represent joint distributions and variable dependency that is both intuitive and comprehensible [18]. Shaded nodes represent observable random variables (In topic modeling these are the words in the documents, visable to the reader); transparent nodes represent unobservable random variables (in a topic modeling context, this would be the latent topics or clusters that describe how the words are generated); arrows the dependencies between those variables. Visualising the model is a substantial first step to developing a true understanding.

Another essential element to understand is variational parameters. As previously alluded to, we wish to capture the inherent randomness or noise in the model. A variational autoencoder trumps a standard autoencoder in capturing this stochastic element of the data [6]. It does this by learning a possible distribution over parameters as apposed to simple parameter estimates. Variational parameters simply learn distributions, not merely fixed values. In the case of variational autoencoders, mean-field approximation is used to learn these parameters [18]. When considering a generative process - as is often the case with autoencoders - a variational approach incorporates another advantage: the difference between estimates is smoothed, allowing for more accurate results when dealing with previously unseen data (this is will be explained at depth later in the dissertation).

A Bayesian approach is applicable as we want to incorporate prior domain knowledge, but how do we approximate variational parameters and/or posteriors in a computationally efficient manor? When performing posterior inference or parameter estimation, complex inter-dependencies between random variables may result in the inability to derive maximum likelihood estimations (MLE), parameter estimates and/or maximum a posterior (MAP) inference [10]. Graphical models describe functions of this form. It is clear from Figure 1 that multiple random variables depend on one another, and as a consequence estimation values of interested are intractable and cannot be solved [11]. Approximation of these values is the best known solution. Thus in deriving meaning from our data and to perform inference accordingly,

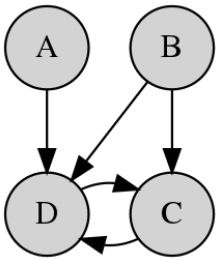

Figure 1: A graphical model example, demonstrating how convoluted variable dependencies can be readily understood through graphical models. The nodes are observable dependent variables, wherein variable dependency is shown through the solid lines [18]

we need to solve for the functional form of the posterior probability of z contingent on x, where

$$\begin{aligned} p(z|x) &= \frac{p(z,x)}{p(x)} \\ &= \frac{p(x|z)p(z)}{p(x)} \end{aligned}$$

However, this proves problematic as for even moderately complex functions $p(x)$ is intractable; calling for approximation of the posterior.

2.1 Variational Inference

Our focal point is to estimate $p(x)$, which can be done in a multitude of ways. Sampling techniques (most commonly Markov Chain Monte Carlo (MCMC) methods) are frequently used - in which a posterior distribution is approximated through sampling in the probabilistic space - however such procedures are computationally expensive; proving infeasible on large datasets [11]. As a consequence computationally cheap systems are growingly attractive.

Variational Inference (VI) is computationally superior as it essentially transforms an iterative process into an optimisation problem. VI attempts to approximate the true posterior distribution with an approximate variational distribution so that $P(Z|X) \approx Q(Z|V) = \Pi Q(Z|V)$ where V is the variational parameter[1]. An algorithm (normally KL divergence) is then used to optimise with respects to v, (normally performed with standard Stochastic Gradient techniques) allowing us to quickly derive an estimation for true posterior $p(z|x)$.

2.2 Auto-Encoders

We want to perform VI on (local) latent variables per observation and maximum likelihood inference on the (global) parameters [10]. An example of an observation is a document. Consider a dataset $X = \{x_i\}_{i=1}^n$. Suppose our data is generated by a two step stochastic process wherein:

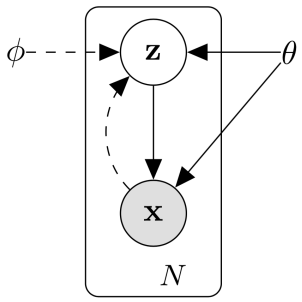

Figure 2: Solid lines representation the generative model. Circles denote random variables, of which those that are shaded are observed variables. Dash lines symbol variational approximation $q_\phi(z|x)$ to the intractable posterior $p_\theta(z|x)$. ϕ and θ are learnt jointly; and denote variational parameters and generative model parameters respectively.

- The prior distribution $p_\theta(z)$ generates the latent variable z_i
- The observation x_i is generated depending on the generated variable z_i, i.e. $p_\theta(x|z)$

We only observe the data $\{x\}_{i=1}^n$ and the latent variables z and generative parameters θ^* are unobserved. The goal is to create an algorithm that is robust enough to work in the case of both intractability and large data [10].

Auto-Encoder Variational Bayes (AEVB) offers a solution to three important problems in a dataset following this structure: (1) estimating the generative parameters θ, which is useful but in the case of identifying θ to investigate the generative process or emulating the hidden generative process to produce artificial data that resembles the real data. (2) Given the parameters Θ, speedy approximate posterior inference on the unobserved z - which can be used for coding. (3) Efficient approximate marginal inference on the observed variable x - which allows for many inference tasks where the prior distribution over x is required [10].

To address this, a recognition model $q_\phi(z|x)$ is proposed as an estimation for the true posterior distribution $p_\theta(z|x)$. The variational parameters ϕ are not derived from closed form expressions [11] and as a result we will design a model to jointly learn ϕ and θ. Note, $q_\phi(z|x)$ can be thought of as a probabilistic *encoder* as it reduces the dimensionality of the dataset by producing a distribution over the latent variables z for the current value of x. Similarly $p_\theta(x|z)$ can be considered a probabilistic *decoder* as it produces a distribution over the datapoint x given the latent variable z[10]. Figure 2 depicts such a model.

2.3 Computation

A growing number of reputable statistical practitioners are moving from frequentist to Bayesian techniques. Among others, probabilistic modeling accommodates this change of approach. Probabilistic methods, as apposed to standard statistical methods, cast variables of interest as distributions of possible

outcomes, as apposed to static parameter estimates. A trivial (but conceptually descriptive) example is linear regression vs probabilistic linear regression. Linear regression finds the best fitting β using OLS or ML, so that $Y = X\beta + \epsilon$. A probabilistic linear regression model then can be written as $Y \sim \mathcal{N}(X\beta, \sigma^2)$. Whilst largely the same as the original linear regression, the second model denotes Y as a distribution. We use MCMC methods or VI to approximate the posterior of all parameters of interest: intercept, x, σ. MCMC encompasses two fundamental benefits:

1. Priors may be included in the formation of the distributions around random variables, making for more informed outcomes and leveraging expert domain knowledge.

2. Quantifying uncertainty within the model, by specifying a posterior distribution for variables (in this example: β) we are thus determining how likely each value of the variable is. As an example, if we have very few data points, our uncertainty for β will be very large, resulting on a platykurtic (large tailed/ large variation) posterior [5].

To summarize, probabilistic programming is novel in that the inputs and outputs are stochastic in some nature.

A powerful probabilistic programming tool, PyMC3 offers an easy to use package for fitting Bayesian models (most frequently MCMC and VI) making it a seamless choice for testing our algorithm [6]. PyMC3 leverages Theano backend - Theano is a Python package that is optimized for manipulating mathematical expressions [25] - for computations - allowing for GPU processing. As well as being well-documented and user-friendly, it's flexibility allow for a myriad of applications.

Specifying a model in PyMC3 is done by collectively allocating a the necessary random variables (RVs) that correspond to important distributions (prior, likelihood etc) and subsequently calling those distributions as methods on the model. This can be further segmented into observed RVs and unobserved RVs, where observed and unobserved RVs are defined via the likelihood $p(x|\theta)$ and prior $p(\theta)$ respectively.

Log-likelihood calculations are frequently used in probabilistic programming as the parameter estimations are the same in that of the regular likelihood however the computation is far simpler. Deterministic transformations (algebraic calculations)are readily stored in an instance of the model class. Interestingly, PyMC3 also performs 'automatic' transformations, to make bounded RVs unbound. This makes sampling increasingly efficient. An example of this given in the official documentation is the log odds transformation of a uniform (bound by 0 and 1) distribution to enable sampling from $\{-\infty, \infty\}$. If required, the specific desired transformation can be specified. It is also possible to explicitly prohibit these automatic-transformations [25].

3 Topic models

Topic modeling is a statistical technique to discover latent topics in text data [2]. It posits many challenges: most of which stem from the vectorisation of text to a numerical values for analysis. Accurately converting text information into numerical matrices without losing information or exceeding a reasonable dimensionality is troublesome. A simple and widely adopted approach consists of:

- Tokenization: create *tokens* by converting sentences into lists of words.
- Removing punctuation, tags and other non language noise.
- Stripping away stop words - which are simple frequently appearing words that add no information like *"the"*, *"and"* etc.
- Stemming words: stripping away prefixes and suffixes to get to the root word (*running* would become *run*).
- Lemmatization: grouping various forms of the same word (example: running, ran) [2].

If performed correctly, *tokens* are then readily converted to binary matrices through one-hot encoding (creating dummy variables) [2].

Some notable terminology (adapted from *Applied Natural Language Processing with Python* [2]):

- Count data: data that can only possess non-negative integer values (as in the number of times a word appears).
- Term-frequency matrix $\underset{i\times j}{TF}$: a matrix mapping the count of specific words within documents - where rows i denote documents; columns j denote terms and the numerical matrix value a_{ij} values correspond to the number of times term j appears in document i.

3.1 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a topic modelling algorithm that clusters unlabelled data.

LDA follows a simple hierarchical formation. A *word* w is a basic unit of discrete data. A *document* $\boldsymbol{W}$ is made up of a sequence of words $\boldsymbol{W} = \{w_1, w_2, ...w_k\}$ and a *corpus* $\boldsymbol{D}$ (which encompasses the entire dataset) is a collection of documents [4].

Our task is to group documents with similar semantic meaning. Human readers are adept at this, however it becomes cumbersome when trying to perform algorithmically. LDA attempts to infer k latent variables (*topics*) for the purpose of categorising documents[4]. The number of bins or 'topics' k is predetermined. With LDA we represent each document as a distribution of topics and each topic as a

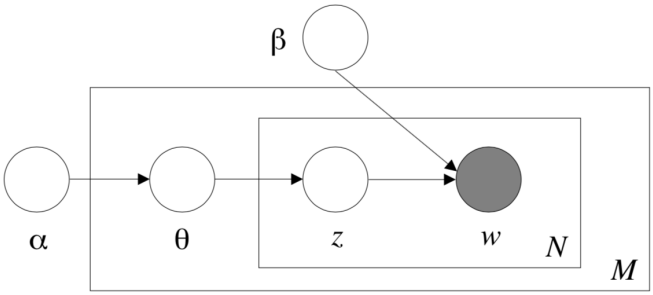

Figure 3: Graphical Representation of LDA parameter dependencies

distribution of words. This method allows a single document to have multiple topics - each represented by a probability in the distribution.

Important to note, although LDA is predominantly used to organise text data, it is not limited to this use case. Any data that follows the relevant structure could be classified with LDA, yet for the purposes of ease of understanding, we keep to the standard unsupervised text use case.

3.1.1 Why use LDA?

LDA boasts a number of advantages to its alternatives. Unigram models are oversimplified (each word in every document is independently drawn form a single multinomial distribution). A mixture of unigram model offers some improvement: it models under the assumption a topic z is chosen and then used to generate N words independently. The downfall of this approach is each document can only be assigned one topic; as apposed to a distribution of multiple topics under LDA. Finally probabilistic latent semantic indexing (pLSI) is another viable topic model, as it does allow for documents to have a number of different topics. However pLSI falls short as each document corresponds to a new topic d and subsequently two issues arise: firstly this overfits on the training data which makes fitting the model to unseen data problematic; secondly additional documents in the training data will result in direct linear growth in model parameters (again increasing the likelihood of overfitting). LDA serves as a clear superior [4].

3.1.2 Assumptions

LDA assumes documents are generated by selecting a topic from a multinomial distribution and subsequently selecting a word - conditional on the topic selection. The fundamental inference of LDA is to calculate the posterior distribution of the latent variables, conditional on the document. A mathematical issue arises as the likelihood function of this posterior distribution is intractable, as a consequence we instead simple approximate. It is apparent that LDA follows our model specifications for AEVB. When approximating, extremely large datasets become problematic for a number of algorithms. Monte Carlo Expectation-Maximisation (and other Sampling based solutions) and batch optimisation are often too

computationally expensive [4]. When dealing with exceedingly large datasets (which is often the case with unlabelled text data) we need to be more computationally efficient; avoiding expensive iterative procedures when approximated. AEVB offer a solution.

4 Inference Methods

4.1 Sampling methods: Markov Chain Monte Carlo

The core intent of Bayesian data analysis is to solve for the posterior distribution

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)}$$

where the denominator is

$$p(X) = \int d\theta^* p(X|\theta^*)p(\theta^*),$$

$p(X|\theta)$ is the likelihood, $p(\theta)$ is the prior and $p(X)$ is the evidence or marginal likelihood [6].

In the case of conjugate priors we are able to arrive at a closed form solution - this is seldom the case in practise. Solving for the denominator is often intractable and consequently we rely on approximation techniques like Variational Inference (VI). A more frequently used alternative to VI is Markov Chain Monte Carlo (MCMC) methods. Although computationally inefficient - these techniques are very powerful [5]. The basic idea is built around random sampling; MCMC methods are best illustrated as a sequence of steps. Firstly, we know that

$$posterior \propto likelihood \times prior$$

this easily obtained function can serve as a target distribution [18]. Then select a proposal distribution to sample from. Drawing samples from this proposal distribution, where each draw depends only on the state of the previous draw, we form a Markov Chain. Further, an acceptance criteria is used to determine which samples we ought to keep [5]. After an indeterminate number of draws - one of the difficulties with MCMC is knowing when to stop iterating - the Markov Chain of correlated draws will converge to a stationary distribution. We are then able to use those samples as draws from the posterior, and subsequently derive functions of the posterior. The key benefit is use that the target distribution only needs to be proportional to the posterior distribution: as ratios are used in the calculation and solving the intractable marginal likelihood is unnecessary.

4.1.1 Metropolis Hastings

The proceeding passage was abstracted from information in *An Introduction to MCMC* [5]. A simple and widely used MCMC technique, Metropolis Hastings, follows a random walk approach. Let $p(x)$ $N(0, \sigma)$

be the proposal distribution and initialize θ at a random value. Set a new proposal value θ_p where $\theta_p = \theta + \Delta\theta$ and $\Delta\theta$ is from the proposal distribution. Calculate the ratio $\rho = \frac{g(\theta_p|X)}{g(\theta|X)}$ (letting g represent the posterior distribution). In the event the proposal distribution is not symmetric, the calculation is weighted - we will focus on the simple case of symmetric proposal distributions. In taking the ratio, the denominators cancel out and as such any $g(x)$ proportional to the posterior is acceptable. As such, we calculate $\rho = \frac{p(X|\theta_p)p(\theta_p)}{p(X|\theta)p(\theta)}$. If $\rho >= 1$ set $\theta = \theta_p$, alternatively set ρ as the acceptance probability. Then draw from a random uniform distribution u (between 0 and 1) and only keep the sample on the condition that $\rho > u$. This works because we always accept a new sample that has a higher probability of being an element of the posterior, however by also accepting samples of a lower probability we prevent our sampler from becoming fixed on a local maximum. Eventually our samplers will converge to a stationary distribution and can be assumed as samples from the posterior.

4.1.2 Gibbs Sampler

Letting $\theta = (\theta_1, \theta_2, ..., \theta_k)$ be a vector of parameters, where we want to approximate the joint posterior distribution $p(\theta|X)$ [6]. Sampling from the joint distribution is very difficult (or impossible), consequently we sample from the conditional distributions of each θ holding all others constant [10].

$$p(\theta_1|\theta_2, ..., \theta_k, X)$$
$$p(\theta_2|\theta_1, ..., \theta_k, X)$$
$$...$$
$$p(\theta_k|\theta_1, ..., \theta_k, X)[5]$$

While sampling θ_i we treat all other parameters as observed. A full iteration is termed one cycle of the Gibbs sampler. The Gibbs sampler can be thought of a special case of the Metropolis Hastings algorithm where we use these conditional probability distributions $p(\theta_i|\theta_1, ..., \theta_k, X)$ as proposal distributions when sampling for the corresponding parameter estimate θ_i [18]. This allows us to accept all samples and is much more efficient than the Metropolis Hasting approach (provided we can derive the conditionals).

4.2 Optimization techniques: Variational Inference

Variational Inference VI focuses on efficiency by introducing a parameterised posterior $q_\theta(z|x)$ [9]. Casting Bayesian Inference as an optimization problem where we choose its parameter θ to maximise a lower bound

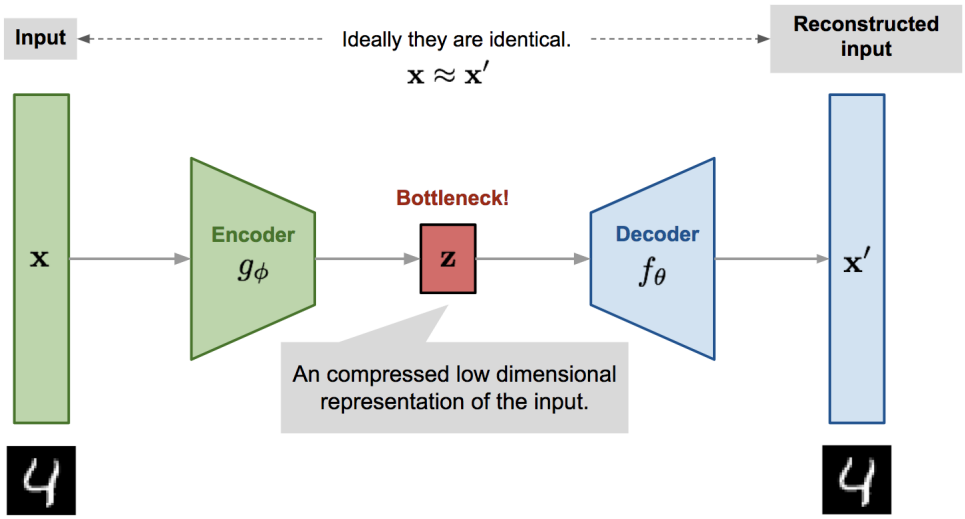

Figure 4: One of the simplest kinds of autoencoders, wherein the goal is to input an image, reduce the dimensionality and then rebuild the original image[15]

$\mathcal{L}$ on the marginal likelihood

$$\log p(x) \geq \log p(x) - D_{KL}(q_\theta(z|x)||p(z|x))$$

$$\mathbb{E}_{q_\theta(z|x)}[\log p(x,z) - \log q_\theta(z|x)] = \mathcal{L}$$

. Since $\log p(x)$ is independent of θ, maximising the bound $\mathcal{L}$ w.r.t θ will minimize the KL-divergence $D_{KL}(q_\theta(z|x)||p(z|x))$ [9]. The estimation $q_\theta(z|x) = p(z|x)$ where $D_{KL} = 0$.

5 Autoencoder

To fully grasp variational autoencoders (VAE), first we may consider more comprehensible autoencoders and subsequently build on from there. Autoencoders are multi-purpose neural networks, best understood by looking at a few cornerstone applications.

5.1 Image reconstructing autoencoder

Figure 4 depicts this conceptually simple autoencoder. Image inputs (X) are fed to the network, our example uses the digit 4 from the famous Mnist dataset [25], an encoder function is then applied g (parameterised by ϕ) to the input variable. We arrive at the bottleneck, appropriately named as data is compressed (in terms of dimensionality) to feed through the bottleneck. Thereafter a decoder function f (parameterised by θ) is applied to the compressed data representation Z for the purpose of closely approximating the original input variable $X^{'}$. θ and ϕ Define the weights and biases associated with the network. Contrary to most statistical models, our only concern with the output vector is to minimize

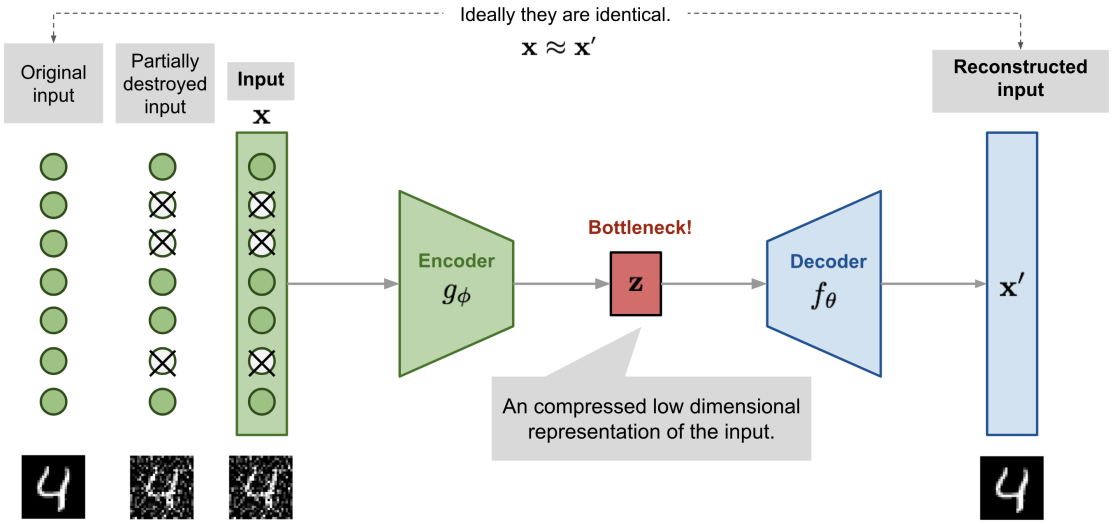

Figure 5: Denoising autoencoders first add noise to data, thereafter deriving the functional relationship between the noisy input $\tilde{X}$ and original input X.[14]

the loss function. $X^{'}$ values are not of interest as at best they simple reproduce the input data. However Z offers more interesting interpretation, since in the instance we produce $X^{'}$ that closely resembles X, we derive a less rich version of the data at Z, without losing any necessary information to sufficiently understand our data set.

By feature engineering they amount of hidden nodes (given a single hidden layer in this neural network) we determine the magnitude to which we reduce data complexity. In the case where the number of hidden layer nodes directly matches that of the input (and by definition output) layers, the network will simply copy the data - demonstrating the trade-off between accuracy and reduced complexity when feature engineering the network.

Notice how the f_θ function provides the generative process in which our data is produced given the latent Z: which perfectly describes our demonstrative example model (LDA). Parameters (ϕ, θ) are jointly learned. A variety of methods are available to quantify the difference between our input vector X and output vector $X^{'}$. If the activation function is sigmoid, we are able to use Cross-entropy to gauge the difference between these vectors. Alternatively a simple MSE loss function can be used:

$$L_{AE}(\phi, \theta) = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - f_\theta(g_\phi(x^{(i)})))^2.$$

$f_\theta(g_\phi(x^{(i)}))$ Is a particularly intuitive loss function as it perfectly maps the process of deriving $X^{'|X}$ since the inner function $g_\phi(x^{(i)}) = Z$ encodes and the outer function $f_\theta(Z)$ decodes the latent variable of interest.

5.2 Denoising autoencoder

Adding some conceptual complexity to the standard autoencoder, a denoising autoencoder Figure (5) adds noise to the input data and thereafter performs the same tasks as a standard autoencoder. Importantly, the cost function:

$$L_{AE}(\phi, \theta) = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - f_\theta(g_\phi(\tilde{x}^{(i)})))^2.$$

minimizes the difference between the noisy input data $\tilde{X}$ and original input X. Noise is added by applying some stochastic masking operation $\tilde{x}^{(i)} \sim \chi(\tilde{x}^{(i)}|x^{(i)})$. This allows for a more generalisable result, since the original autoencoder allows for overfitting as we are fit a the exact X to itself (this concern is amplified when then number of network parameters exceeds the number of data points). Adding noise is equivalent to adding variation, providing more reliable, robust, results [22].

Human observers can readily identify occluded images. The network has to discover relationships between dimensions of X to make accurate inference provided there is missing information [22]. This added level of variation provides for learning more robust latent representations as it is often necessary to learn combinations of many input dimensions.

5.3 Variational autoencoder (VAE)

More accurately a derivation of VI, variational Bayes and graphical models - the task of a VAE is to map the input to a distribution, not a fixed vector X (as the aforementioned autoencoders do) [9]. As a consequence, the once static encoder and decoder are now *probabilistic* (Figure **??**). The latent representation of the data set z is now a distribution of possible outcomes as apposed to static values. We assume the data is generated by:

- z is generated from a prior distribution $p_{\theta^*}(z)$
- x is generated from a conditional distribution $p_{\theta^*}(x|z)$

The true parameters $\theta*$ and latent variables Z are unknown and need to be approximated. Parameters ϕ and θ are learned jointly from probabilistic encoder $q_\phi(z|x)$ and probabilistic decoder $p_\theta(x|z)$ jointly. Per iteration: once $q_\phi(z|x)$ is approximated (through back-propagation), samples $z \sim q_\phi(z|x)$ are drawn and fed into the generator network to produce $x' \sim p_\theta(x|z)$ Thereafter x and x' compared to calculate the loss **??**.

The resulting autoencoder allows for new data generation, as a random sample can be taken from our latent distribution z and fed into the probabilistic decoder to generate a reasonable output. This new data generating produces realistic results with VAEs because they map the space between input data smoothly; whereas other autoencoders segment input nodes more aggressively and as such random noise

fed into a standard decoder will produce random noise. The difference is due to the core assumption that the random noise fed into the decoder network is from a distribution similar to that of the training data. *see appendix for the derivation of the loss function.*

5.3.1 Optimization of the loss function

Recall the optimal values θ^* and ϕ^* are found by $\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathcal{L}(\theta, \phi; x)$ [10]. In Variational Bayesian methods, this loss function is known as the *variational lower bound* or the *evidence lower bound - ELBO* - due to the fact that KL-divergence is non-negative, we can conclude from equation 8 that $\mathcal{L}(\theta, \phi; x)$ is the lower bound of $\log p_\theta(x)$.

$$\mathcal{L}(\theta, \phi; x) \leq \log p_\theta(x)$$

Thus, minimizing the loss is maximizing the lower bound of the probability of generating real data samples.

5.3.2 The reparametrization trick

We employ alternate optimization to find the desired values of θ^* ϕ^*, in which we:

- Find an initial value for θ^1 solving for the derivative of the loss w.r.t θ, thus: $\theta^1 = \nabla \arg\min_{\theta,\phi} \mathcal{L}(\theta, \phi; x)$ holding ϕ constant.
- Then taking this result as a constant input and deriving an expression for $\phi^1 = \nabla \arg\min_{\theta,\phi} \mathcal{L}(\theta^1, \phi^1; x)$
- Thereafter using ϕ^1 as an input for the calculation of θ^2 and so on for an indeterminate 'long' period of time until our current approximate values for θ^i and ϕ^i converge to stable parameter values that can be inferred as close to θ^* and ϕ^*.

Probability density functions $Q_\theta(Z|X)$ and $P_\phi(X|Z)$ are learned jointly - implying the optimization is calculated w.r.t both θ and ϕ jointly. Given the loss function 7 and assuming both: the special case of a zero mean unit variance Gaussian prior and $\exp \Sigma_\phi(x)$ is used in place of $\Sigma_\phi(x)$; our loss function $\mathcal{L}(\theta, \phi) = -\mathbb{E}_{z\sim q_\phi(z|x)}(\log p_\theta(x|z)) + \frac{1}{2}\sum_K [\exp \Sigma_\phi(x) + \mu_\phi^2(x) - 1 - \Sigma_\phi(x)]$ it can be shown that θ^i is easily found by:

$$\begin{aligned}
\theta^i &= \nabla_\theta \{-\mathbb{E}_{z\sim q_\phi(z|x)}(\log p_\theta(x|z)) + \frac{1}{2}\sum_K [\exp(\Sigma_\phi(x)) + \mu_\phi^2(x) - 1 - \Sigma_\phi(x)]\} \\
&= -\mathbb{E}_{z\sim q_\phi(z|x)}(\nabla_\theta \log p_\theta(x|z)) \\
&= \frac{1}{L}\sum_{\ell=1}^{L} \log p_\theta(x|z^\ell)))
\end{aligned}$$

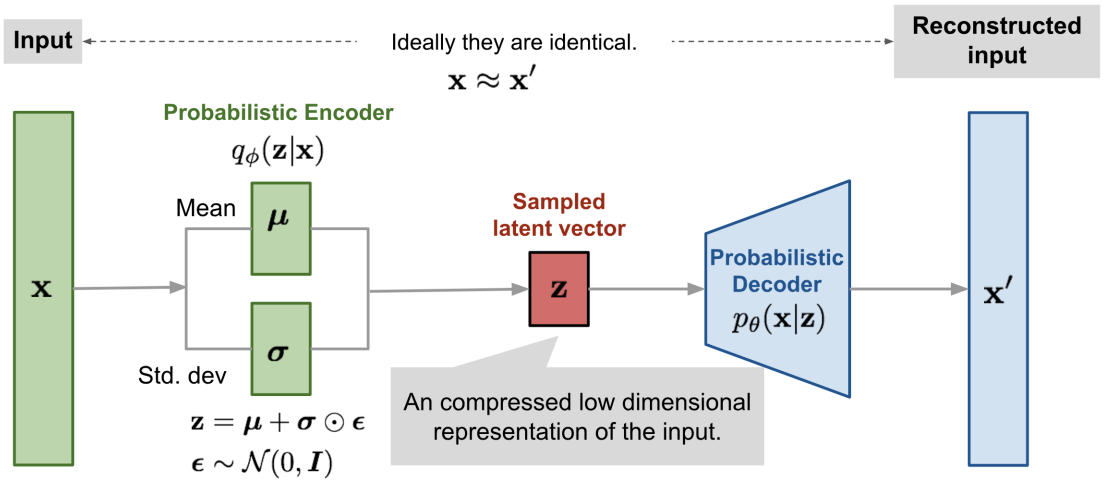

Figure 6: Variational autoencoder (VAE) [16]

.

where $z^\ell \sim Q_\phi(Z|X)$ and $\frac{1}{L}\sum_{\ell=1}^{L}$ is the Monte Carlo estimate for the expectation operator. This results relies on the fact that ∇ can move inside of the expectation operator.

The computation of the second step (finding the derivative of ϕ^i) becomes problematic, since:

$$\phi^i = \nabla_\phi\{-\mathbb{E}_{z\sim q_\phi(z|x)}(\log p_\theta(x|z)) + \frac{1}{2}\sum_{K}[\exp(\Sigma_\phi(x)) + \mu_\phi^2(x) - 1 - \Sigma_\phi(x)]\}$$

A problem arises as the ϕ appears in the distribution w.r.t the expectation taken ($\mathbb{E}_{z\sim q_\phi(z|x)}$) and as such $\nabla_\phi \mathbb{E}_{z\sim q_\phi(z|x)}(\log p_\theta(x|z)) \neq \mathbb{E}_{z\sim q_\phi(z|x)}(\nabla_\phi \log p_\theta(x|z))$. Suppose z is a continuous random variable such that $z \sim q_\phi(z|x)$. Using some auxiliary variable ϵ with some independent marginal likelihood $p(\epsilon)$ we are often able express z as some deterministic variable $z \sim g_\phi(\epsilon, x)$ [10]. Thus allowing us to reparametrization the problematic expectation to a functional form that is differentiable w.r.t ϕ.

5.3.3 Kullback-Liebler divergence

Kullback-Liebler divergence (for brevity, known as KL-divergence) measures the degree to which two probability distributions differ. Consider the probabilities density functions $p(x)$ and $q(x)$, KL-divergence is derived as:

$$D_{KL}(p||q) \triangleq \int p(x) \log \frac{p(x)}{q(x)} \partial x$$

In the event where we log these densities, we derive KL-divergence as:

$$D_{KL}(p||q) = \sum_{x=1}^{n} p(X = x) \log \frac{p(X = x)}{q(X = x)}$$
$$D_{KL}(p||q) = \sum_{x=1}^{n} p(x) \log \frac{p(x)}{q(x)} \tag{1}$$

The divergence satisfies three properties, the divergent properties [7]:

1. Self similarity: $D_{KL}(p||p) = 0$
2. Self indication: $D_{KL}(p||q) = 0$ only if $p = q$
3. Positivity: $D_{KL}(p||q) \neq 0 \ \forall p, q$

6 Automatic differentiation variational inference (ADVI)

ADVI is a VI system that alleviates statisticians/data scientists from the iterative constraints of probabilistic modeling. The cumbersome nature of fitting convoluted models to large datasets prove both mathematically and computationally arduous. The scientist simply feeds ADVI the dataset and probabilistic model, thereafter ADVI automates the derivation of an efficient VI algorithm [12]. Importantly, in our case, this extends to a multitude of models: conjugate priors are not a prerequisite.

Where probabilistic programming is the procedure of describing a probabilistic model as a computer program and compiling that program into an efficient inference executable [12]; automatic inference receives probabilistic code as an input and outputs a well-ordered algorithm for the computation of the model. There have been copious automatic inference modules and packages, however the vast majority of which rely on expensive MCMC iterations, whereas ADVI leverages the speed of a VI approach. Although VI boasts a number of advantages, it non-the-less requires laborious model derivations - the derivations of joint (prior and likelihood) distributions [12]. Thus, ADVI enables the automation of these VI derivations, by taking a dataset and a probabilistic model as an input and returning posterior inference about the models latent variables [12].

More formally, ADVI:

1. In order to define all possible latent variables on the same space, enabling the use of a single variational family for all models, ADVI transforms the model to one with unconstrained real-valued latent variables [12]. From $p(x, z)$ to $p(x, \zeta)$ where z is a representation of latent variables and ζ is the unconstrained real-valued latent variable. Essentially, removing the restrictions that bound z. Thus, by making this transformation a corresponding VI solution can be defined as $D_{KL}(q(\zeta)||p(\zeta|x))$, resulting in all latent variables being defined on the same space.

2. The problematic gradient - of the log joint distribution w.r.t the latent variables $\nabla_z \log p(x, z)$ - of the objective function is subsequently recast as an expectation over q (the reparameterization trick aforementioned). This reparameterization allows for the implementation of Monte Carlo approximation methods [12].

3. The gradient is then reparameterized i.t.o a standard Gaussian by performing a transformation within the variational family [12]. Which, in turn, allows for efficient MCMC sampling - only sampling from a standard Normal distribution.

4. The variational distribution is optimized by deploying noisy gradients, whilst adaptive tuning step-size sequence provides speedy parameter convergence.

6.1 Leveraging a variational auto-encoder and ADVI

Taking a document as input, the encoder calculates the Gaussian parameters unconstrained real coordinate space. More generally, the encoder calculates the parameters of the transformed latent variables. The encoder then outputs the variations (distribution over) mean and standard deviation - with the previously explained $2 \times (K-1)$ parameters. Batches are used as input, as apposed to the entire training data, for rapid updating. The output is two matrices (variational mean z_μ and variational standard deviation z_{std}) each with dimensions $batchsize \times (K-1)$. In ADVI, z_{std} is defined $p = \log(\exp(std) - 1)$ - bounded to be positive. Rearranging yields $std = \log(1 + \exp(p))$, which is numerically stable (in that erroneous inputs have a lessor effect on outputs). A Bernoulli corruption process is applied to input documents with the aim of improved generalizability. Empirically, however, it does not appear to better results. The auto-encoders output is then fed to the variational parameters θ. The autoencoders parameters are then optimized using ADVI. (how do I reference the notebook?).

The approximate variational parameters are used to draw samples from the variational posterior - in our case, the posterior mean of the β word-topic distribution. Performance can be gauged by examining the most frequently appearing words for each topic and assessing if clear groups have arisen. When comparing the most frequent words per topic with that of scikit-learn implementation (which leverages online stochastic variational inference[8]) the results appear comparable. Importantly, AEVB-ADVI rely on mean-fields approximation, a much simpler instance than the standard scikit-learn implementation which relies on conjugate priors.

7 Understanding the algorithm holistically

For the purpose of providing a clear understanding of how these mechanisms are combined, this section outlines the systematic sequential process followed to when grouping the above topics and applying AEVB

with ADVI on LDA. The successive nature of the process make the imagining of alternative applications easy, as different models may be easily substituted for one another.

7.1 LDA preprocessing for VAE modeling

LDA simply provides a comprehensible use-case of a model that fits the required structure to apply AEVB. Assuming we want to learn the latent variable K (number of topics), the data preparation is as follows:

1. A corpus of documents $\mathcal{D}$ is stripped to numerical data by. This is done by binarily encoding a term frequency matrix $\underset{D\times V}{tf}$; where each row D_i represents each document and each column V_i each word. For reduced redundancy the vocabulary is limited to the most frequent v words. Notable, for large datasets the tf matrix is often extremely sparse (the vast majority of entries are zero).

2. The documents are split into training and test sets to improve the odds of successively obtaining generalisable results.

3. Solving the log-likelihood of an LDA model with w tokens and K topics requires two matrices. $\underset{D\times K}{\theta}$ describing the topic distribution over documents and $\underset{K\times V}{\beta}$ portraying the word distribution over topics.

4. Under standard LDA the random variables θ and ϕ are drawn from the Dirichlet distribution, however when utilizing ADVI all latent variables are transformed onto the unconstrained real co-ordinate space [12]. A Dirichlet hosts parameters on a simplex, so the new dimensions are the original dimensions -1, thus $K-1$. To perform this mapping, PyMC3 uses a technique called a 'centred-stick-breaking' transformation. To streamline the approximation process we transform to a much simpler distribution - allowing speedy sampling - as such mean-field approximation is used: the variational posterior on the transformed parameters is represented as a spherical Gaussian. The number of variation parameters is then $2\times(K-1)$ (mean and standard deviation).

8 VI and MCMC hybrid: combining inference methodologies

The most frequently used algorithmic solutions to Bayesian posterior approximation $p(z|x)=\frac{p(z)p(x|z}{\int p(z)p(x|z)dz}$ (namely, the posterior is the ratio of the prior times the likelihood to the marginal likelihood) are MCMC and VI, the former boasting the advantages of being nonparametric and asymptotically exact whilst the latter maximises an explicit objective function (normally resulting in faster approximation) [9].

The general consensus is that machine learning specialists must trade-off approximation techniques in order to prioritise either speed or accuracy [9]. Diederik P. Kingma and Max Welling [9] offer an interest

proposal, wherein practitioners integrate one (or more) MCMC methods into the VI approximation - potentially closing the speed/accuracy differential between methodologies.

MCMC begins by randomly drawing some value Z_0 from a set distribution $q(Z_0)$ or $q(Z_0|x)$, thereafter apply a *stochastic transition operator* (that denotes the probability of accepting a move to a new position in the chain) to the random draw [9]

$$z_t \sim q(z_t|z_{t-1}, x).$$

Eventually we arrive at z_T: a random variable that converges its distribution to the true posterior [9]. Problematically, we do not know when we have completed enough iterations. Further, a single iteration can be extremely expensive if we are dealing with exorbitant amounts of data.

Using a stochastic Markov chains $q(z|x)$, we set $y = \{z_0, z_1, z_2, ..., z_{T-1}\}$ as *auxiliary random variables* [9]. By integrating these auxiliary variables into the variational lower bound, we derive:

$$\begin{aligned}
&\mathcal{L}_{aux} \\
&= \quad \mathbb{E}_{q,z_T|x}[\log[p(x, z_T)r(y|x, z_T)] - \log q(y, z_T|x)] \\
&= \quad \mathcal{L} - \mathbb{E}_{z_T|x}\{D_{KL}[q(y|z_T, x)||r(y|z_T, x)]\} \\
&\leq \quad \mathcal{L} \leq \log[p(x)][9].
\end{aligned}$$

The best estimation of the marginal posterior is specified as $q(z_T|x) = \int q(y, z_T|x)dx$. $r(y|x, z_t)$ represents a free to specify auxiliary inference distribution [9]. $q(z_T|x)$ now equates $q(z_T|x, y)$ and as such the auxiliary variable may assist in more accurately approximating the true posterior. Selecting $r(y|x, z_T)$ proves problematic as the best theoretical choice $q(y|x, z_T)$ is frequently intractable. Although there are many ways to set $r(y|x, z_T)$, the *Autoencoding variational Bayes* paper focuses on the initializing the function follows a Markov Chain structure $r(z_0, ..., z_{T-1}|x, z_T) = \prod_{t=1}^{T} r_t(z_{t-1}|x, z_t)$, resulting in the variational lower bound taking the form

$$\begin{aligned}
\log p(x) \quad &\geq \quad \mathbb{E}_q[\log p(x, z_T) - \log q(z_0, ..., z_T|x) \\
&+ \quad \log r(z_0, ..., z_{t-1}|x, z_T)] \\
&= \quad \mathbb{E}_q[\log[p(x, z_T)/q(z_0|x)] \\
&+ \quad \Sigma_{t=1}^{T} \log r_t(z_{t-1}|x_t, z_t)/q_t(z_t|x, z_{t-1})].
\end{aligned}$$

Note the subscripts on q_t (the transition operator) and r_t (inverse models) is indicative of allowing for multiple functions (at different points in the Markov Chain) [9]. Further, allowing for additional optimization of equation by defining q_t and r_t as some flexible parametric form [9].

The variational lowerbound is thus approximated by incorporating sampling processes into variational approximation [9]. This can be invoked at different stages of the estimation process, sampling for different variables, allowing flexibility in the degree of inclusion [9].

9 Application

Latent Dirichlet Allocation has become is used so frequently that a number of different computational algorithms have been applied in training - each with a unique set of advantages and limitations. As with many Bayesian inference problems: exact solutions are intractable and estimation algorithms are required to parameter estimation [6]. Bayesian inference is simply inferring meaning from data in a probabilistic manner - incorporating prior knowledge when learning a posterior distribution [6]. Based on Bayes theorem:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$$

where $p(\theta|x)$ is the posterior distribution (how probable each parameter value is given the data and incorporating the prior knowledge); $p(x|\theta)$ is the likelihood - how well the data fits each parameter value; $p(\theta)$ describes the prior (allows domain expertise and outside knowledge to be included in the model); and $p(x)$ denotes the evidence, the probability distribution of the observed data independent from any parameter value [18].

The prior and likelihood are frequently known, however evidence proves problematic. The evidence is computed as $p(x) = \int_\theta p(x|\theta)p(\theta)$ (a normalization factor [6]) - this expression is frequently intractable in higher dimensions [4]. When applied to LDA, the posterior represents the probability of a topic conditional on a document.

Two frequently used methods include a sampling approach: Markov Chain Monte Carlo (MCMC); and approximation an approach: Variational Inference (VI) [18]. MCMC techniques aim to generate samples from the true posterior - by forming a Markov Chain that's stationary distribution is equivalent to the posterior [6]. Both the Metropolis-Hasting and Gibbs Sampling algorithms are dependent on the mathematical property of *reversibility*.

A Gibbs sampler relies on the computation of conditional probabilities. Suppose the Markov Chain to be specified has D dimensions: the joint probability density function may be intractable, however the conditional density function of each dimension is not, thus by setting initial values and iteratively updating each conditional θ_i value after a substantially long period the samples should converge to the true parameter θ_i values (and hence be samples from the true posterior) [6]. Not distribution is assumed, and with sufficient computation the true posterior is guaranteed to emerge. Problematically, however, it

is unclear how long the algorithm will take to converge, and as such the computational need for complex models with large datasets is often beyond reach.

Alternatively, variational inference (VI) offers a simplified solutions that trades accuracy for efficiency. VI rests on the premise that a simpler distribution can be used to approximate the true posterior and consequently the problem is turned into an optimization problem - finding the parameters that maximise the approximate distribution [18]. Given a family of distributions, VI searches for the distribution that most accurately represents the true posterior: by defining a parameterised family of distributions and thereafter optimizing over the parameters allows one to find the closest parameter to the target distribution - on the condition that the error is well defined [6]. It is imperative to note VI assumes the distribution is of the chosen parameterised family of distributions - which inherently implies bias [1].

In most cases the Gibbs sampler is more accurate (without bias) but possesses a larger variation and thus requiring vastly more computation, whilst VI is exceedingly faster - far more scalable - but biased. The more restricted (simpler) the assumed family under VI, the faster the computation but the larger the bias [6].

The auto-encoder offers an potential alternative computation framework. Simply a variational neural network wherein the response variable is a replica of independent variable - essentially mapping the information to a lower dimensional space. The structure of the neural network allows for speedy computation which could possibly allow us to scale to much larger datasets.

9.1 Experimental design

A number of experiments were conducted in aid of answering the follow research questions:

1. *Does the autoencoder LDA provide comparable results - and predictive capability - to the well establish Scikit-learn - online learning - LDA?*
2. *Is it possible to scale the autoencoder to large datasets?*
3. *Does the autoencoder offer significant processing-efficiency advantages as datasets scale?*

All algorithms were implemented in Python [1] and are publicly available on my personal website [2] as well as in the appendix.

9.2 Implementation

Before investigating the scalability of the autoencoder, first we need to ensure adequate results are achieved by the model. In comparing the autoencoder to more conventional algorithmic techniques a collection of computation packagers were utilized. All algorithms were implemented in Python.

[1] `https://www.python.org`
[2] `http://www.zachwolpe.com/research/`

1. Autoencoder: The autoencoder is specified with the aid of Pymc3 [21] - a probabilistic programming framework built on Theano [24] computational backend.

2. Variational Inference: Scikit-Learn [20] offers an optimized online variational Bayes (VI) implementation of LDA - which will be used to draw a comparison to gauge the autoencoders performance.

3. Computational setup: Experiments were implemented on a 2012 Macbook Air with a 1.8 GHz Intel Core i5 processor with 4GB memory.

9.3 Dataset

Initial experimentation is done on the *'20 NewsGroups'* dataset[3] - a dataset on which the autoencoder is known to deliver satisfactory performance. Thereafter, if comparable results are achieved, the autoencoder is to be tested on a range of dataset and corpus sizes.

The 20 Newsgroups dataset is a collection of newsgroup documents almost evenly partitioned into 20 newsgroups. The original dataset contains approximately 20'000 documents, however the Sklearn packaged version (used here) contains $11'314$. This first experiment is intentionally done on a manageable corpus - if comparable results are reached, the two models should be contrasted on larger datasets - possibly further segregating testing into large vs small document size. A corpus was creating containing the $w = 1000$ most commonly appearing words.

9.4 Data preprocessing

An imperative part of the process of performing statistical/analytical procedures on non-numerical data types is to convert the data to a numerical structure: the difficulty in which is to convert the data in such a manor to balance capturing all existing information without an excessively verbose data-structure.

Since LDA is primarily concerned with the semantic meaning of a document processing is done under the intuitive assumption that both words in varying forms ("running" and "run") and synonyms ("better" and "greater") can be grouped. For LDA we want to covert the raw data to a BOW (bag-of-words) matrix: each row representing a document; each column a word and each cell the count - number of times word w appears in document d. The text processing procedure follows a fairly generic NLP (natural language programming) sequence [3]:

1. Stopwords: removal of all words that are abundant throughout all written documents but offer no meaning, words like: *that, the, and* etc. A stopword list was downloaded from Sklearn [20].

2. Stemming: aggressively simplify similar word for processing such that many varying forms are reduced to the stem of the word form: *said*, *says* and *say* all shorted to something like *sai* [3].

[3] `http://people.csail.mit.edu/jrennie/20Newsgroups/`

Stemming is often unnecessary, too aggressive and can detract from the meaning of the corpus [3].

3. Lemmatization: reduce words such that varying forms of the same word are reduced to a single representation (*running* and *runner* both becoming *run*. Grouping words of analogous meaning (for example: *fantastic* and *brilliant*) as to reduce word-count and maintain information. This can be further extended to writing grouping word pairings or triplets are are frequently found throughout the corpus. This extension is, however, often superfluous.

4. Tokenization: The documents of processed words are then converted into lists of strings, with each unique word added to a aggregated list encompassing the entire vocabulary.

5. Vocabulary pruning: the BOW matrix is sparse by design, but excessive sparsity is undesirable and as such the vocabulary list ought to be reduced to a reasonable length (reasonable with respect to the total corpus size). There are no clear metrics on which to base this decision and as such experimentation and intuition play a prime role in optimizing this word count choice. Superabundant words (appearing a number of documents above a threshold) are disregarded as they offer no purposeful distinction between different documents. Excessively infrequent words are also discarded for the same purpose. Thereafter the w most common words are kept. $w = 1000$ is a somewhat shallow choice for this dataset, however the online variational Bayes implementation works very well for $K = 20$ topics; which is known to be the true structure of the corpus and as such $w = 1000$ is a reasonable choice.

6. BOW: the bag-of-words is then readily computed by aggregating the frequency of each remaining word in each document.

Consequently the corpus of raw text documents has been converted a numerical equivalent whilst minimizing entropy and without ballooning dimensionality.

9.5 Model architecture

9.5.1 Priors

Both the θ and β priors are uniformly distributed Dirichlet-multinomial distributions where the probability of each of the K nodes is $\frac{1}{K}$

$$\beta \sim Dir_K(\frac{1}{K}); \theta \sim Dir_K(\frac{1}{K})$$

which is, intuitively, an obvious choice as one would not want to bias the learning algorithm in any way.

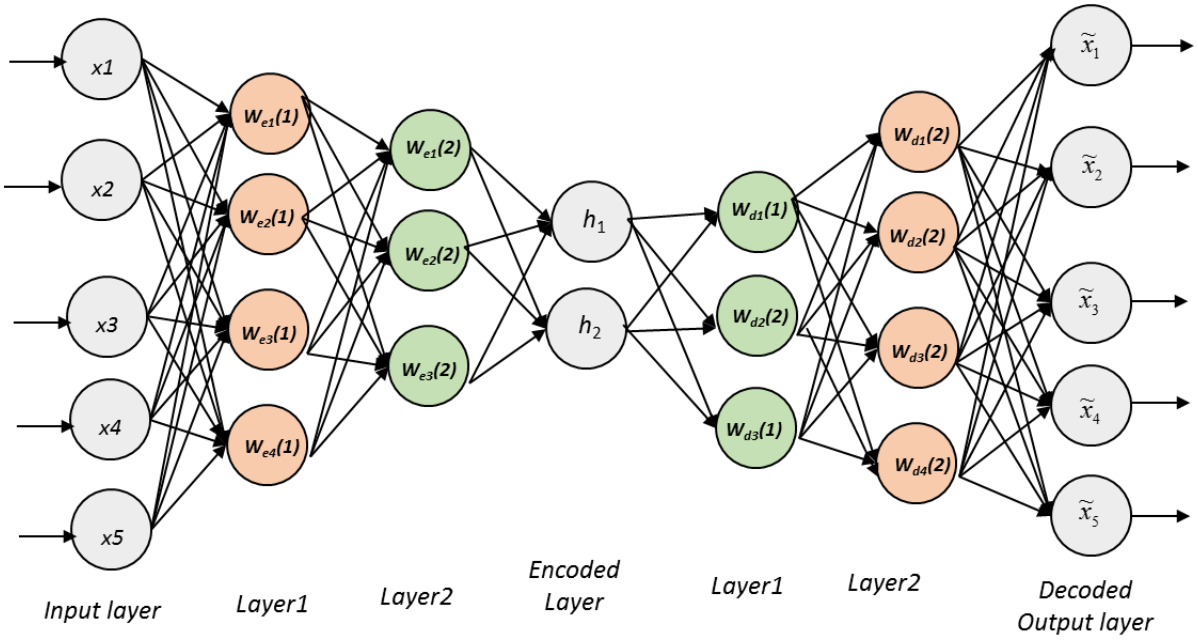

Figure 7: Autoencoder architecture

9.5.2 Neural network architecture

The Dirichlet priors follow a K dimensional simplex - where K is the number of topics - and as such each distribution leverages $K-1$ dimensions [6]. In training a probabilistic model, distribution are learned as apposed to simple values: in our case mean-field approximation is used to learn spherical Gaussian parameters, thus $2 \times (K-1)$ dimensions are required. Thus the autoencoder is structured such that $2 \times (K-1)$ nodes are encoded. 128 documents are passed into the encoder per batch, which is then reduced to the first hidden layer consisting of W nodes (W is the number of words in the vocabulary). Thereafter the second hidden layer consists of $2 \times (K-1)$ nodes - meeting the requirements of the spherical Gaussian structure within the K dimensional simplex. Thus the parameters learned in the simplex correspond to topic allocation. This structure is repeated 100 times - forming 100 hidden layers and thus indicative of deep learning [6].

9.6 Topic model coherence

Topic model performance is at the epicenter of addressing the research questions at hand, however topics model evaluation offers vastly more nuance and subjectivity than traditional modeling techniques [6]. This is a consequence of the nature of the data (written text).

An initial (and undoubtedly important) step in the topic model evaluation process is simply to inspect the n (normally 10 is sufficient) most common words in each topic. If a model has found logical word groupings to represent topics, several natural underlying themes will be self-evident. Although an essential starting point, extensive model evaluation calls for numerical inference.

Quantifying text-based model performance is inherently difficult by nature. Two humans could derive very different meanings from the same passage; and consequently pick out very different key themes. Most methods thus simplify the analysis by focusing on objective quantities [18].

One method of deriving topic significance scores leverages computing topic similarities and dissimilarities (KL divergence, cosine, and correlation) however topic significance scores rely on complicated functions with free parameters [13]. Resulting in unnecessary complications and the possibility of overfitting [13].

9.6.1 Topic coherence

Topic coherence is a prominent technique of evaluating topic model performance. Coherence attempts to quantify how logical (coherent) topics are by measuring the conditional probability of words given a topic. All flavours of topic coherence follow the general equation [17]:

$$\text{Coherence} = \sum_{i<j} score(w_i, w_j)$$

Where words $W = w_1, w_2, ..., w_n$ are ordered from most to least frequently appearing in the topic. The two leading coherence algorithms (UMass and UCI) essentially measure the same thing [19] and as such I have chosen to focus on UMass. The UMass *scores* between $\{w_i, w_j\}$ combinations (which are summed subsequent to calculation) are computed as:

$$score^{k}_{UMass}(w_i, w_j|K) = \log \frac{D(w_i, w_j) + \epsilon}{D(w_i)}$$

Where K_i is i^{th} topic returned by the model and w_i is more common than w_j $(i < j)$. $D(w_i)$ is the probability of a word w_i is in a document (the number of times w_i appears in a document divided by total documents). $D(w_i, w_j)$ is the conditional probability that w_j will occur in a document, given w_i is in the document - which eludes to some sort of dependency between key words within a topic [17]. ϵ simply provides a smoothing parameter, which is often simply set to 1 to avoid taking log(0) in the case where the conditional probability is zero.

One concern arises, as it is overtly clear that the number of w_i, w_j combinations balloons to absurd quantities for even even relatively small corpus' with few words in the aggregated vocabulary.

As such, it is adequate to simply compute the outer connections on Figure 8: computing coherence for word pairs $\{(w_1, w_2), (w_2, w_3), ..., (w_{n-1}, w_n)\}$ [17].

To improve the reliability of computed coherence, we can factor in the sampling distribution of documents; ultimately learning the spread of topic coherence for a given dataset learning K soft clusters. This is achieved by:

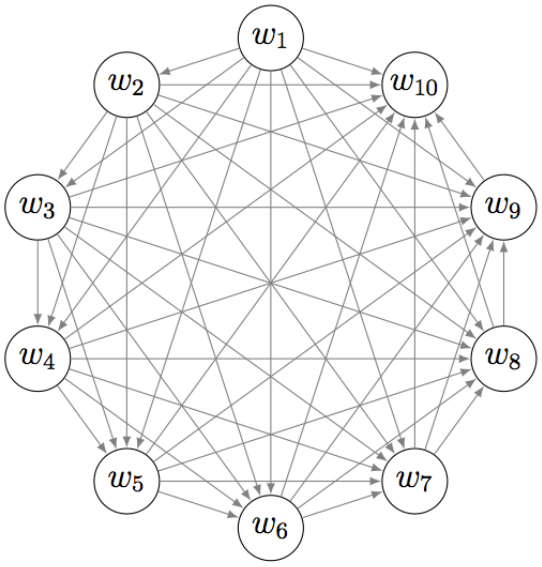

Figure 8: A graphical representation of how the number of word pairings balloons as words of significance are added to the analysis [23]

algorithm algpseudocode

Algorithm 1 Topic coherence around sampling distribution

Result: Topic model coherence spread for varying topic counts
initialization
  for k over all values of K **do**
    for i in $1 : t$ **do**
      Randomize train-test split.
      Compute the LDA parameters for both models
      compute coherence on the returned topics
    end for
  end for

The results are indicative of the spread of coherence scores at each topic level. The optimal number of topics to learn for the dataset in question is that which produces the highest coherence score. Different models (in our case, the autoencoder and Sklearn online VI LDA) are readily contrasted as higher coherence is indicative of a superior model. The empirical results of Algorithm 1 are depicted in Figure 9, indicating the sampling spread over varying topic sizes. As scores are relative and higher scores are favourable it is clear that the online VI approach offers superior results.

9.7 Topic Alignment

Since LDA is an unsupervised learning technique, its learned soft clusters are not labeled and as such cluster-meaning needs to be inferred by human judgement. Another issue concerning model comparison is that clusters are not learnt in the same order, topic (cluster) 1 of model A does not necessarily represent topic (cluster) 1 of model B [6]. Thus before $Topic_1^a$ vs $Topic_1^b$ comparisons can be done, topics need to be aligned such that $Topic_1^a = Topic_1^b$.

Again, sorting by laboriously checking all permutations is either extremely expensive or outright impossible. The Hungarian algorithm offers an elegant way of matching up similar topics [23]. This algorithm resolves a problem known as the *standard alignment problem.* In the context of topic modeling, after model parameters have be learnt, a *cost matrix* is created wherein the rows represent the topics of model A and the columns the topics of model B. Thus if the diagonals of the *cost matrix* are highly correlated, the topics are aligned. Suppose there are K topics and as such the *cost matrix* (denoted C) has dimensions $\underset{k\times k}{C}$. The Hungarian algorithm sorts matrix C such that the diagonals are correlated and thus topics are aligned.

Finally, aligned topics are to be viewed and assessed by human judgement. Similar topics should be present.

9.7.1 Topic coherence

After specifying the vocabulary size $W = 1000$ and processing the dataset into a term-frequency matrix, the process of training the models on various plausible values of K topics (and again on various train-test-splits) was conducted, resulting in the coherence scores in figure 9.

An interesting pattern seems to emerge as the coherence scores of both models are fairly matched at 5 topics - a model specification that we can assume to be over simplistic given the known - 20 groups - structure of the dataset. Whilst this is the best performing autoencoder model, it is the worst performing Sklearn model. The subsequent autoencoder models appear to degrade with complexity (number of topics). The Sklearn output is at its apex (however slight) at the logical $K = 20$, where the number of topics is consistent with the known structure of the dataset. It may - however - be possible that the marginal difference between the models is not as significant as Figure 9 suggests. As such, further analysis will consider two comparisons: that of 5 and of 20 topics. 5 as it is the smallest discrepancy between models and highest performing autoencoder model; and 20 as it offers a comparison of a larger performance gap between models as well as being the highest performing Sklearn model.

9.7.2 Topics alignment

Aligning topics allows one to contrast topic correlation as well as inspect the most common words in aid of gauging the logical connection between topics. Figure 10 provides the result of aligning models of various topic specifications. The *average correlation* (on the diagonals) peaks at 0.55 and steadily decreases as models grow in complexity. The first and second image - per row - indicates the cost matrix before and after alignment respectively; whilst the third image provide the distribution of correlations (skewing to lower values as models grow in complexity). Models where $K > 10$ appear fairly independent, again questioning the equivalence of each solution.

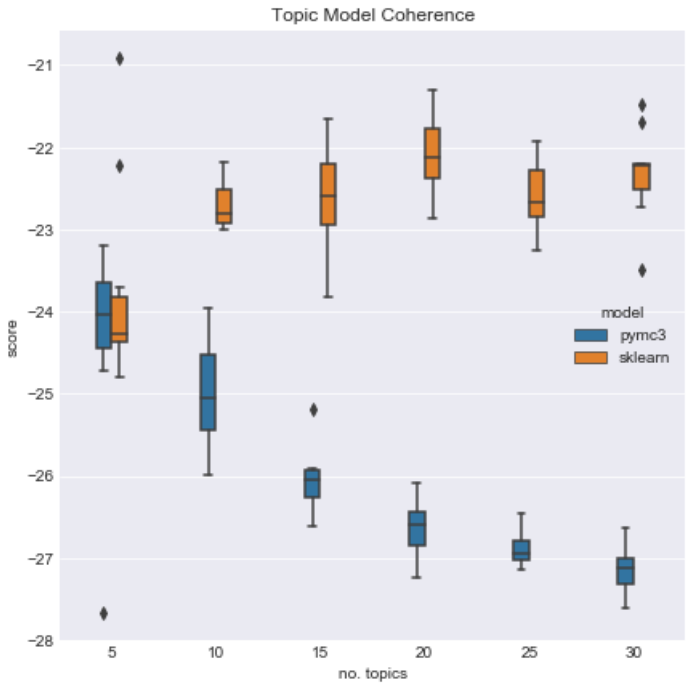

Figure 9: Topic model coherence achieved for various numbers of topics, contrasting VI LDA and autoencoder variational Bayes LDA implementation.

9.7.3 Topic inspection

Now that topics have been aligned, the next step is to inspect the most common words in each topic, provided in Figures 11, 12 and 13.

Nearing identical results are produced by the simplest ($K = 5$) models. It is not overtly obvious what the semantic association with each topics is, however there are lucid, differentiable work groups. Topic 4 appears the most comprehensible - pertaining to *data* and *information*; topic 3 capturing numerical values. Whilst topics 0 and 2 do not offer readily interpretable results - they do seem to capture unique words and thus serve as reasonable clusters. Topic 1 appears to capture odd characters strings that are partially integers, partially characters. The corresponding topics between models have an average correlation of 0.55, which is intuitively confirmed by the apparent consistency in the output.

Consider now, the $K = 10$ topic models. Correlation between respective topics in the autoencoder LDA and Sklearn LDA significantly deteriorates - falling to 0.43. The models do, however, appear fairly matched: producing comparable common-word per topic output in figure 12. Thereafter, as model complexity increases (growing value of K) the autoencoder's performance steadily declines. The additional topics estimated appear to be duplicates of the original smaller topics (provided in figure 13. One possibility is that the repetitive topics may be indicative of in an ill-sized corpus vocabulary - however this is indefensible as the Sklearn implementation appears to produce sound results (producing its best performance at $K = 20$ topics). The correlation between models falls to 0.23 for $K = 20$ - again indicative of

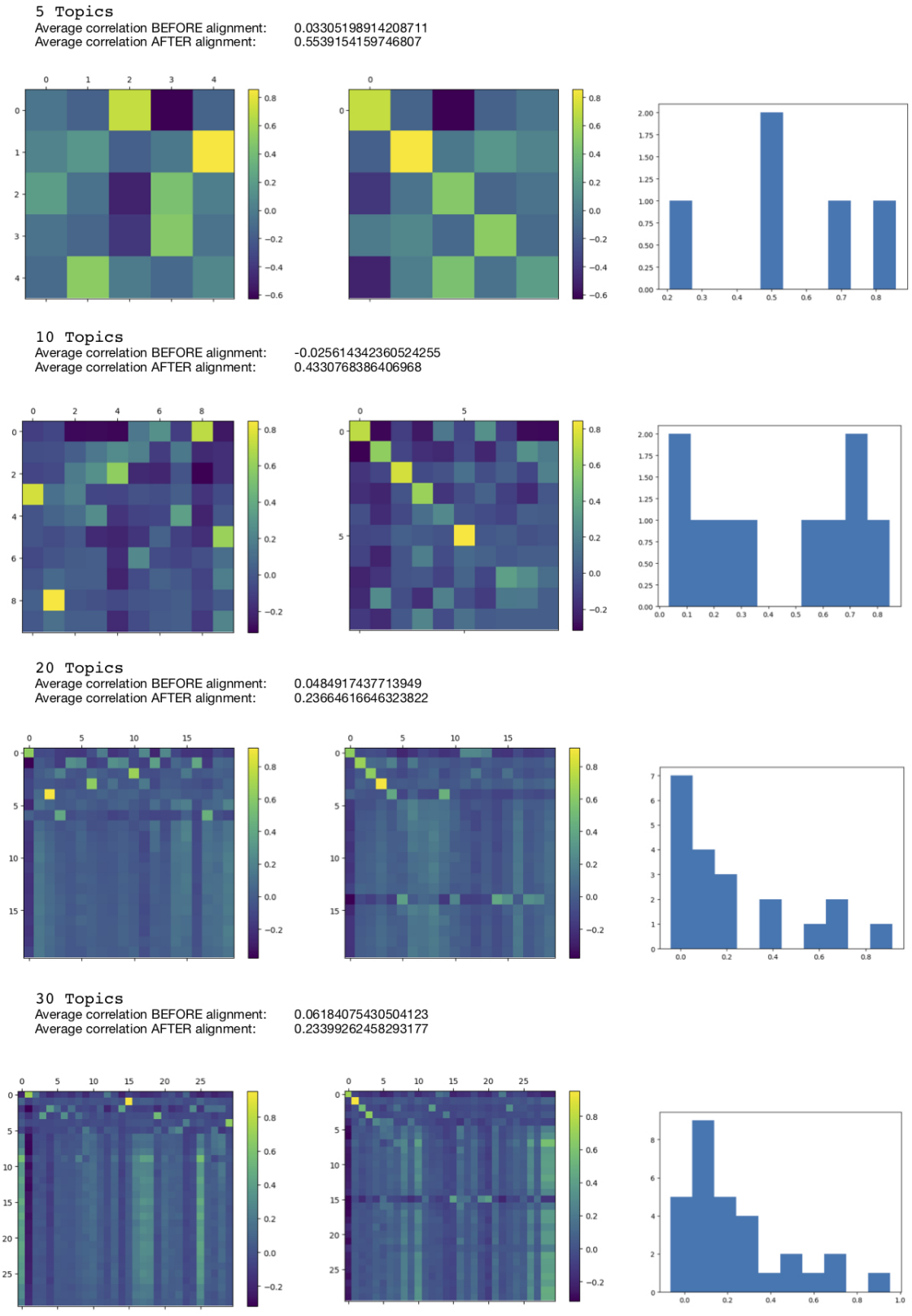

Figure 10: The cost matrix for different K values, before and after topic alignment. The average correlation on the diagonals offers insight into how well the topics of been aligned - supported by the distribution of correlations graph (figure 3 of each row).

```
5 Topics

Pymc3:
Topic #0: people don just think like god know time say does
Topic #1: ax max g9v b8f a86 75u 1t pl bhj 34u
Topic #2: use just like drive key know don does chip bit
Topic #3: 10 00 game team 15 11 20 year 12 play
Topic #4: edu file use program com information space mail windows available

Sklearn:
Topic #0: people don think just like know time say did said
Topic #1: ax max b8f a86 145 1d9 g9v pl 0t 34u
Topic #2: edu use com like windows thanks file does know software
Topic #3: 10 25 20 14 15 17 11 16 12 13
Topic #4: key use program space information available public government number data
```

Figure 11: The most common words in of each $K = 5$ topics, post Hungarian alignment.

```
10 Topics

Pymc3:
Topic #0: people god don think just like know time say said
Topic #1: windows drive use card like does problem disk thanks know
Topic #2: 00 10 11 15 20 12 17 55 16 18
Topic #3: just don good like year car game think team time
Topic #4: don use just like time need make know government think
Topic #5: ax max g9v a86 b8f 75u 1t 145 bhj 3t
Topic #6: like just water don know new used good insurance use
Topic #7: edu db thanks know mail com cs like does don
Topic #8: file edu use key information program space available data ftp
Topic #9: just don like know good does think new make ve

Sklearn:
Topic #0: people god don think just say does believe know time
Topic #1: file windows use available drive window output version disk data
Topic #2: 25 10 14 17 16 15 11 12 20 24
Topic #3: don think like just good know ve time game year
Topic #4: key government public use states president security law encryption health
Topic #5: ax max b8f a86 145 1d9 g9v pl 0t 34u
Topic #6: said people armenian armenians israel did went children turkish war
Topic #7: edu com mac apple pub university ca cs gov mail
Topic #8: use like know just thanks program don help work using
Topic #9: space new price address 1993 card used send car 800
```

Figure 12: The most common words in of each $K = 10$ topics, post Hungarian alignment.

```
20 Topics

Pymc3:
Topic #0: people god don think just know like say time said
Topic #1: 00 10 11 20 55 15 12 17 16 25
Topic #2: space file information nasa gun new launch 1993 university data
Topic #3: ax max g9v 75u a86 b8f 1t bhj 3t 145
Topic #4: just like don good car year time game think team
Topic #5: like know don just good does new think edu thanks
Topic #6: know like just don good does edu thanks new think
Topic #7: know like don just good does edu new think thanks
Topic #8: know like just don does edu good think thanks new
Topic #9: like just know don does good edu new think thanks
Topic #10: just like know don does good edu thanks new think
Topic #11: like know don just edu does good think new com
Topic #12: don like know just does edu good need make got
Topic #13: don like just know edu does good actually did interested
Topic #14: use edu windows file program drive key using does card
Topic #15: like know just don good does thanks new edu think
Topic #16: like know don just does good edu think thanks new
Topic #17: like just know don does good edu new thanks think
Topic #18: like just know don does edu good new thanks make
Topic #19: know like just don does good thanks new edu think

Sklearn:
Topic #0: don people know just think like say did time god
Topic #1: 17 10 w7 14 11 27 24 25 16 13
Topic #2: space 1993 april health 20 10 year new center washington
Topic #3: ax max b8f a86 145 1d9 g9v pl 0t 34u
Topic #4: year team game play players season league teams vs hockey
Topic #5: bit key card know thanks does chip like serial memory
Topic #6: 15 price 10 power new 12 sale offer sound 16
Topic #7: output program info rules source section read int define build
Topic #8: edu com send cs ca motif gov sun request mit
Topic #9: car good like just got games bike runs run hit
Topic #10: problem use problems just work image don error know using
Topic #11: war israel jews israeli jewish south world peace anti history
Topic #12: government people law president state public states new think administration
Topic #13: armenian armenians turkish greek turkey turks armenia killed russian people
Topic #14: file windows entry ms files dos version os drivers comp
Topic #15: available data use information software ftp 800 server info pub
Topic #16: bus soon theory says local land vga water video sense
Topic #17: drive scsi 25 hard disk drives 32 tape controller hardware
Topic #18: window graphics application package display user code 3d widget event
Topic #19: mac pc 92 picture mb edu pub 91 word language
```

Figure 13: The most common words in of each $K = 20$ topics, post Hungarian alignment.

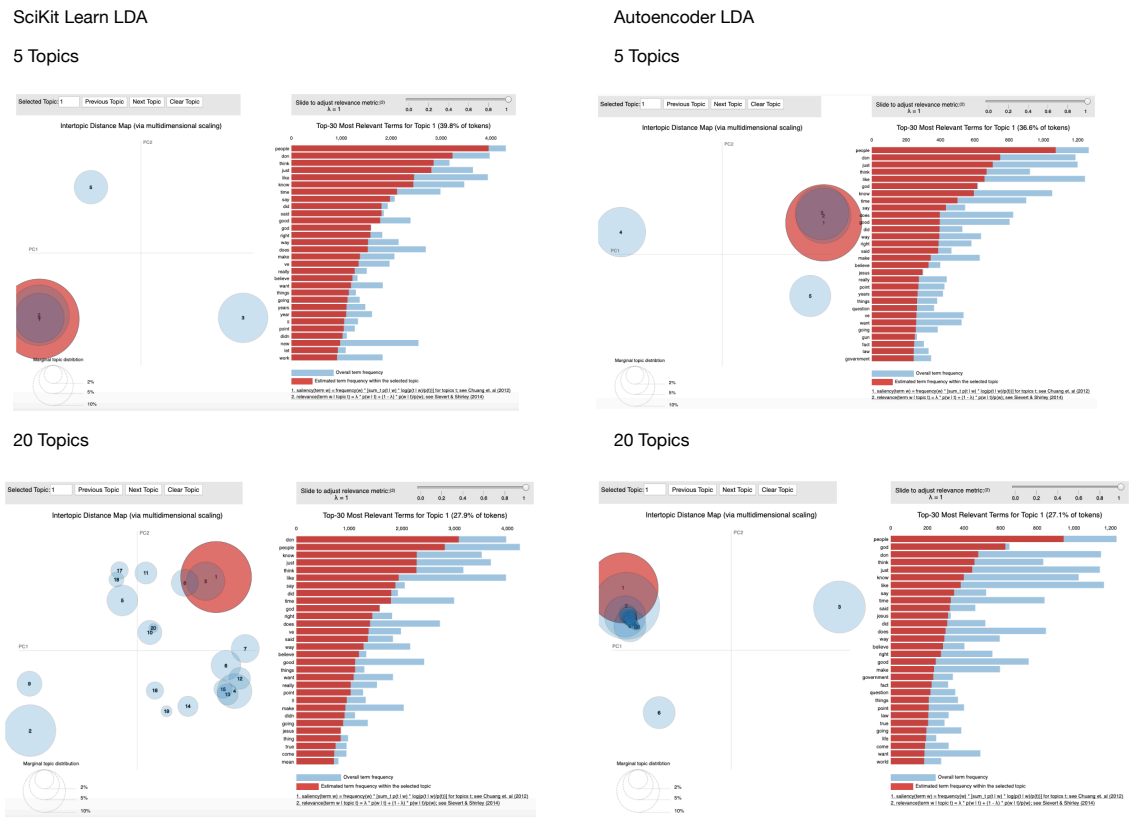

Figure 14: Visualization of trained LDA models for various specifications of K.

inconsistent results.

9.7.4 Topic visualization

Finally, visualizing the model topic provides a clear depiction of the independence (or lack thereof) - providing a pragmatic tool when assess the success of clustering. Figure 14 provides a clear graphical representation of:

- How independent (strong) topics are - how spread-out the circles are.
- How elaborate topics are - the size of each circle.
- The key words in each topic.
- The likelihood that a particular key word occurs in a topic.
- The predicted (red) vs actual (blue) occurrence of a word in a topic.

Inspecting the two 5 topic models supports the above-mentioned claim both models produce analogous results: clusters are of comparable size, spacing and word distributions. Again, supporting the argument above, when both models are trained on 20 topics the SciKit learn model reigns superior - offering clusters throughout the graph as apposed to the autoencoders heavy nested groups.

9.8 Understanding the autoencoders inadequate performance

It is abundantly clear the auto-encoder fails to deliver analogous results. To further investigate what mechanisms impede the autoencoder, consider it's structural makeup: A probabilistic neural network wherein the response variable is the independent variable and as such attempts to map the same information into a compressed -lower dimensional- space. Probabilistic in the sense that the encoded data is represented as a distribution (a Gaussian) and as such the moments of the distribution are learnt as apposed to simple numerical values for the weights and biases. The weights and biases of the autoencoder are randomized at 0.5 and 0 as to be as neutral as possible: being random starting values that bare no consequence on the end results.

9.8.1 Understanding performance flaws

To assess the poor performance of the autoencoder, analysis is conducted on the early stages of parameter tuning/weight updating. It is plausible the autoencoder:

- Biases towards certain topics, not sufficiently exploring the space of possibilities.
- Overemphasizes the importance of the initial random conditions, making the conditional probability of a topic selection in $p(Epoch_{t+1}|Epoch_t)$ unjustifiably high.

The following experiment was designed to addresses these concern:
Step 1:

1. Write a callback function that is executed per epoch.
2. On execution, perform two actions: sample the current $\underset{(\text{Doc}\times\text{Topic})}{\theta}$ posterior distribution both once and $10'000$ times.
3. Cease after 100 epochs as the development of the θ probabilities should be readily examinable.

Step 2:
The collected data needs be restructured, as the sequence collects data in order of occurrence (Epoch to Document to Topic) however a simpler way to analyse the format would constitute (Document to Epoch to Topic) as the goal is to investigate the distribution of topic allocation over the iterative process.
Step 3:
The reordered data is then plotted and the topic allocation per epoch is investigated. It is possible that certain topics are dismissed after inadequate sample representation in the initial - random - conditions.

Figure 15 depicts the result for the $10'000$ posterior samples per epoch - starting from epoch 1 after one full iteration as epoch 0 is simply the uniform simplex prior distribution. It appears inherently clear

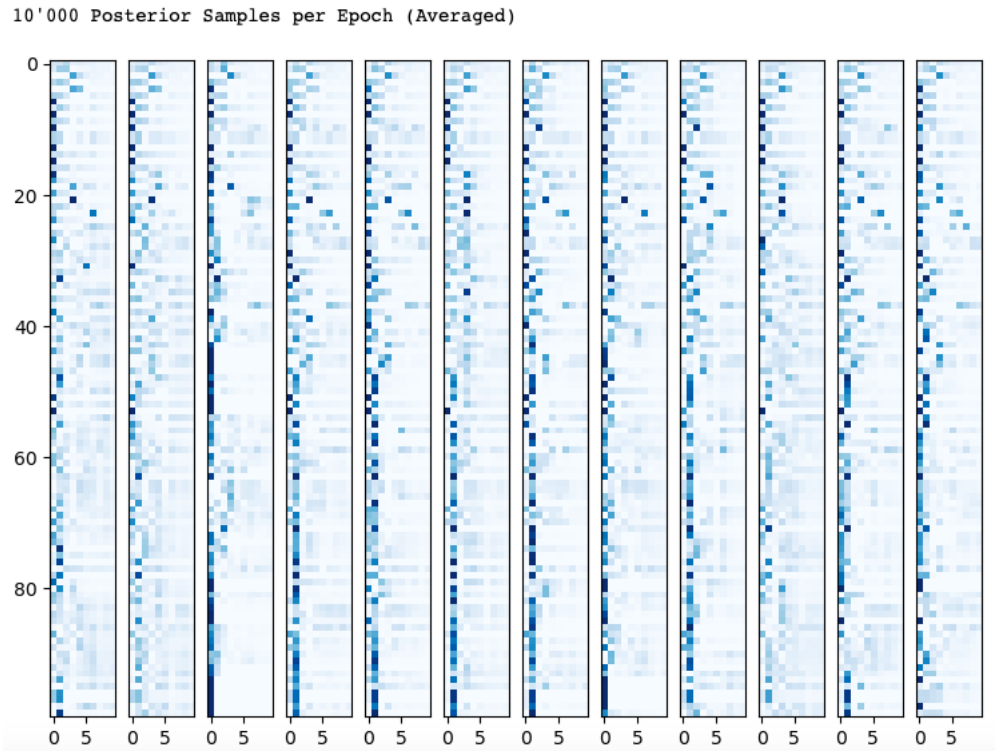

Figure 15: 12 documents (of the 128 used for the first training batch) are randomly selected. Each column represents a documents - ranging vertically from $Epoch_1$ to $Epoch_{100}$. Each column ranges horizontally from $Topic_0$ to $Topic_10$ - since 10 topics are learnt in this model specification.

that topic allocation is heavily skewed towards topics $K \in (0, 1, 2)$ - even in the very early stages of training. Assessing an individual document is futile as this behaviour would be warranted in the case where a single document is dominated by one topic and quickly allocated as such. One would expect a more uniform distribution initially and thereafter topics to be pruned until the true topics emerge. Empirically the first few topics are arbitrarily allotted to nearly all documents. The same results can be expected in the case of $K = 20$ topics as the model fails to utilize a substantial portion domain space.

Summing the intermittent θ probabilities over a number of epochs allows for the empirical discover of the distribution of topic allocation. Figure 16 depicts these results. Since the underlying structure of the dataset is known - $K = 20$ - one would expect a more uniform allocation over a random section of topics. The computation of the empirical distribution is repeated to account for the sampling distribution. The overwhelming skewed distribution is consistent with the prior findings. To further account for sampling distribution - and support the claim of unsolicited topic allocation bias - the topic distribution is assessed over 5 topics: where one would expect a far more random/uniform distribution.

Recall the callback written not only collects a $10'000$ sample posterior, but also a single sample from the intermediate posterior. These results are summarised in figure 17 and are expected to be totally random (at least for the first few epochs). Whilst significantly more distributed that the more elaborate, the results are disproportionately skewed towards the the first few topics.

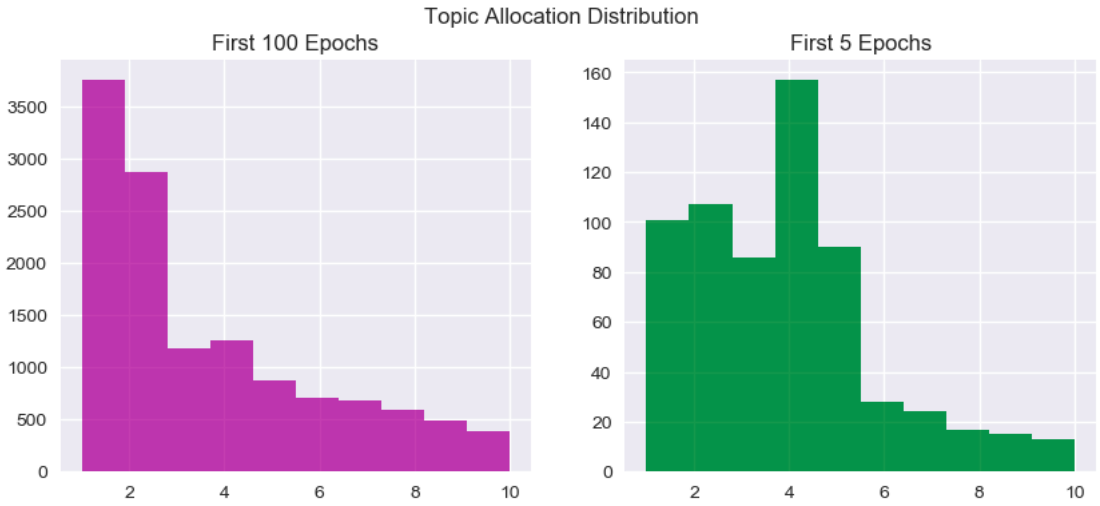

Figure 16: Distribution of topic allocation over 100 epochs.

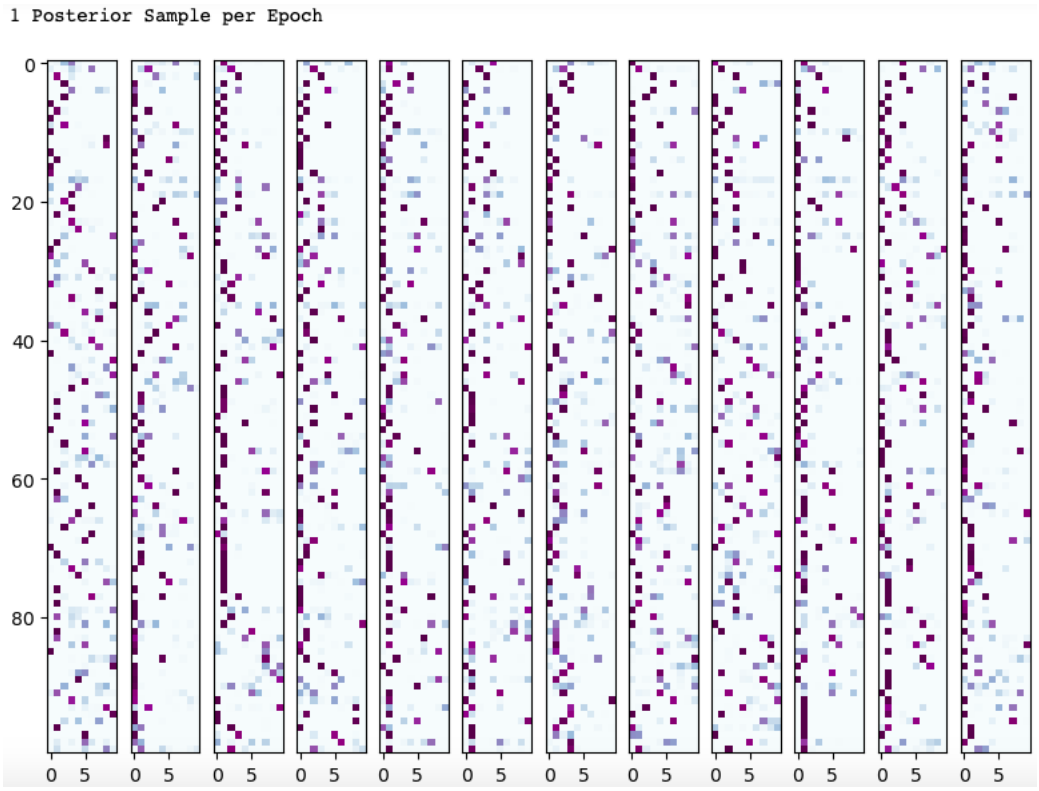

Figure 17: A single θ sample per epoch.

10 Conclusion

An autoencoder variational Bayes latent Dirichlet allocation model boasts the theoretical advantage of brisk computation, however this allure evades comparisons of quality. In conducting this analysis my contribution is two fold:

- perform a thorough comparison between autoencoder LDA and VI LDA, uncovering the benefits and drawback of the autoencoder as a alternative brisk approach.
- Uncover the autoencoder's shortcomings in an attempt to deduce bias that limits the autoencoder's performance - in addressing *why* the autoencoder fails.

After detailed experimentation it is readily apparent that the autoencoder falls short when juxtaposed against established, well engineered, statistical techniques. When analyzing the topic coherence performance the autoencoder offers analogous results to the VI LDA for simple models, however as models grow in complexity it is unambiguously inferior to established statistical methods - accounting for sampling variability. Results are only comparable under textbook conditions. We show that the encoder fails to adequately explore the domain spaces and heavily biases initial random conditions.

Performance may be improved by changing the specified prior or neural network architecture. Different priors may influence the explored search space. Other datasets should also be examined to determine the auto-encoders performance for various corpus sized and complexities. All code is available on my personal website [4].

[4] `http://www.zachwolpe.com/research/`

References

[1] Hagai Attias. A Variational Bayesian Framework for Graphical Models. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 209–215. MIT Press, 2000.

[2] T. Beysolow. *Applied Natural Language Processing with Python: Implementing Machine Learning and Deep Learning Algorithms for Natural Language Processing.* Apress, 2018.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python.* O'Reilly Media, Inc., 1st edition, 2009.

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[5] Petros Dellaportas and Gareth O. Roberts. *An Introduction to MCMC*, pages 1–41. Springer New York, New York, NY, 2003.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] J. R. Hershey and P. A. Olsen. Approximating the Kullback Leibler Divergence Between Gaussian Mixture Models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 4, pages IV–317–IV–320, April 2007.

[8] Matthew Hoffman, Francis R. Bach, and David M. Blei. Online Learning for Latent Dirichlet Allocation. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 856–864. Curran Associates, Inc., 2010.

[9] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, December 2013.

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[11] Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David Blei. Automatic Variational Inference in Stan. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 568–576. Curran Associates, Inc., 2015.

[12] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic differentiation variational inference. *J. Mach. Learn. Res.*, 18(1):430–474, January 2017.

[13] Wei Li and Andrew McCallum. Pachinko allocation: Dag-structured mixture models of topic correlations. In *Proceedings of the 23rd international conference on Machine learning*, pages 577–584. ACM, 2006.

[14] Lilian Weng. Denoising autoencoder, 2018. [Online; accessed March 22, 2019].

[15] Lilian Weng. Image reconstructing autoencoder, 2018. [Online; accessed March 22, 2019].

[16] Lilian Weng. variational autoencoder model with the multivariate Gaussian assumption, 2018. [Online; accessed March 22, 2019].

[17] David Mimno, Hanna M Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. Optimizing semantic coherence in topic models. In *Proceedings of the conference on empirical methods in natural language processing*, pages 262–272. Association for Computational Linguistics, 2011.

[18] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013.

[19] David Newman, Jey Han Lau, Karl Grieser, and Timothy Baldwin. Automatic evaluation of topic coherence. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 100–108. Association for Computational Linguistics, 2010.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using PyMC3. *PeerJ Computer Science*, 2:e55, apr 2016.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[23] Keith Stevens, Philip Kegelmeyer, David Andrzejewski, and David Buttler. Exploring topic coherence over many models and many topics. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 952–961. Association for Computational Linguistics, 2012.

[24] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[25] Jake VanderPlas. *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media, Inc., 1st edition, 2016.

11 Appendix

11.1 Derivation of KL-divergence

Now suppose $p(x)$ and $q(x)$ are two multivariate Gaussian distributions where (Σ notation used over σ^2 as consider vector space and not individual values):

$$p(x) \sim \mathcal{N}(x; \mu_1, \Sigma_1)$$
$$q(x) \sim \mathcal{N}(x; \mu_2, \Sigma_2)$$

then our multivariate normal density (with k dimensions: $\{x_1, x_2, ...x_k\}$) is defined as (note the double bars portray the determinant and not the absolute value):

$$N(x; \mu, \Sigma) = p(x) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\left[-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right]$$

thus if we $\log p(x)$:

$$\log p(x) = -\frac{K}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_1| - \frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1)$$

similarly:

$$\log q(x) = -\frac{K}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_2| - \frac{1}{2}(x-\mu_2)^T \Sigma_2^{-1}(x-\mu_2)$$

and thus the KL-divergence between is defined as:

$$\begin{aligned}
D_{KL}(p||q) &= \sum_{x=1}^{n} p(x) \log \frac{p(x)}{q(x)} \\
&= \sum_{x=1}^{n} p(x)\{\log p(x) - \log q(x)\} \\
&= \sum_{x=1}^{n} p(x)\{-\frac{K}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_1| - \frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) + \frac{K}{2}\log(2\pi) + \frac{1}{2}\log|\Sigma_2| \\
&+ \frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)\} \\
&= \sum_{x=1}^{n} p(x)\{\frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} + \frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2) - \frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\}. \qquad (2)
\end{aligned}$$

Further the latter two expressions can be simplified using some identities of linear algebra, where Tr denotes the *Trace* (summation of the diagonal elements) of a square matrix and $\sum p(x)X = \mathbb{E}X$:

$$
\begin{aligned}
&\sum_{x=1}^{n} p(x)\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) \\
&= \mathbb{E}_p[\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)] \\
&= \mathbb{E}_p[Tr(\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1))] \\
&= \mathbb{E}_p[Tr(\frac{1}{2}(x-\mu_1)(x-\mu_1)^T\Sigma_1^{-1})] \\
&= Tr(\mathbb{E}_p[(x-\mu_1)(x-\mu_1)^T]\frac{1}{2}\Sigma_1^{-1}) \\
&= Tr(\Sigma_1\frac{1}{2}\Sigma_1^{-1}) \\
&= \frac{1}{2}Tr(\mathrm{I}_K) \\
&= \frac{1}{2}K
\end{aligned}
\tag{3}
$$

where $\mathbb{E}_p[(x-\mu_1)(x-\mu_1)^T]$ is the covariance matrix thus $\mathbb{E}[(x-\mu_1)(x-\mu_1)^T] = \Sigma_1$. Secondly:

$$
\begin{aligned}
&\sum_{x=1}^{n} p(x)[\frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)] \\
&= \sum_{x=1}^{n} p(x)[\frac{1}{2}[(x-\mu_1)+(\mu_1-\mu_2)]^T\Sigma_2^{-1}[(x-\mu_1)+(\mu_1-\mu_2)]] \\
&= \sum_{x=1}^{n} p(x)[\frac{1}{2}(x-\mu_1)^T\Sigma_2^{-1}(x-\mu_1) + \frac{2}{2}(x-\mu_1)^T\Sigma_2^{-1}(\mu_1-\mu_2) + \frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2)] \\
&= \mathbb{E}_p[\frac{1}{2}(x-\mu_1)^T\Sigma_2^{-1}(x-\mu_1) + \frac{2}{2}(x-\mu_1)^T\Sigma_2^{-1}(\mu_1-\mu_2) + \frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2)] \\
&= \mathbb{E}_p[\frac{1}{2}(x-\mu_1)^T\Sigma_2^{-1}(x-\mu_1)] + \mathbb{E}_p[\frac{2}{2}(x-\mu_1)^T\Sigma_2^{-1}(\mu_1-\mu_2)] + \mathbb{E}_p[\frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2)]
\end{aligned}
\tag{4}
$$

where the first expression is similar to equation 3 thus $\mathbb{E}_p[\frac{1}{2}(x-\mu_1)^T\Sigma_2^{-1}(x-\mu_1)] = Tr(\Sigma_2^{-1}\Sigma_1)$; the second expression $\mathbb{E}_p[\frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2)]$ is a constant thus $\mathbb{E}_p[\frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2)] = \frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2)$ and the third expression $\mathbb{E}_p[\frac{2}{2}(x-\mu_1)^T\Sigma_2^{-1}(\mu_1-\mu_2)] = 0$

Thus substituting the results from **??** and 3 into 2 we obtain the final KL-divergence equation:

$$
D_{KL}(p||q) = \frac{1}{2}[\log\frac{|\Sigma_2|}{|\Sigma_1|} - K + Tr(\Sigma_2^{-1}\Sigma_1) + (\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2)] \tag{5}
$$

11.2 Derivation of the loss function

The intractable computation of $p(z|x)$ calls for approximation of a tractable close substitute $q(z|x)$. Variational Inference is employed to resolve this, as VI poses an inference problem as an optimization problem. We model $p(z|x)$ with $q(z|x)$ where $q(z|x)$ has a simple distribution - in our case: Gaussian. Thus if we minimize the difference between the two distributions, by definition q will approximate p - a

perfect application of KL divergence. Given Bayes theorem: $p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$:

$$\begin{aligned} D_{KL}[q_\phi(z|x)||p_\theta(z|x)] &= \sum_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p_\theta(z|x)] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p_\theta(x|z) + \log p_\theta(x) - \log p_\theta(z)] \end{aligned}$$

the expectation is over z, this $p_\theta(x)$ can be removed as a constant w.r.t the z:

$$\begin{aligned} &D_{KL}[q_\phi(z|x)||p_\theta(z|x)] - \log p_\theta(x) = \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p_\theta(x|z) - \log p_\theta(z)] \\ &\log p_\theta(x) - D_{KL}[q_\phi(z|x)||p_\theta(z|x)] = \mathbb{E}_{z\sim q_\phi(z|x)}(\log p_\theta(x|z)) - \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p_\theta(z)] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}(\log p_\theta(x|z)) - D_{KL}[q_\phi(z|x)||p_\theta(z)]. \end{aligned} \tag{6}$$

The VAE objective function consists of two distinct parts. Term 1 in equation represents the reconstruction likelihood, whereas term 2 ensures that the learned distribution $q_\phi(z|x)$ is sufficiently close to the prior distribution $p_\theta(z)$. The loss function is simple the negative of the objective function:

$$\mathcal{L}(\theta, \phi) = -\mathbb{E}_{z\sim q_\phi(z|x)}(\log p_\theta(x|z)) + D_{KL}[q_\phi(z|x)||p_\theta(z)] \tag{7}$$

similarly:

$$\log p_\theta(x) - D_{KL}[q_\phi(z|x)||p_\theta(z|x)] = -\mathcal{L}(\theta, \phi). \tag{8}$$

Notice, the presents of $D_{KL}[q_\phi(z|x)||p_\theta(z|x)]$ provides a built in regularizer (a term that enables smoothing that prevents overfitting). $\log p_\theta(x)$ is the reconstruction loss term. The optimal parameter values θ^* and ϕ^* are found by minimizing the loss function $\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathcal{L}(\theta, \phi)$ where ϕ and θ represent the weights for the generative and reconstruction network parameters respectively.

The term $\mathbb{E}_{z\sim q_\phi(z|x)}(\log p_\theta(x|z))$ is also referred to as the log likelihood because it the log of the density will give the square error between the mean and the data sample, for example if we assume $p_\theta(x|z) \sim \mathcal{N}(\mu_\theta(z), \Sigma_\theta(z))$, then the density is given as $p_\theta(x|z) = \frac{1}{\sqrt{(2\pi)^k|\Sigma_\theta(z)|}} \exp \frac{(x-\mu_\theta(z))^T \Sigma_\theta^{-1}(z)(x-\mu_\theta(z))}{2}$; then $\log p_\theta(x|z) \propto (x-\mu_\theta(z))^T \Sigma_\theta^{-1}(z)(x-\mu_\theta(z))$. $(x-\mu_\theta(z))$ is a reconstruction (sum of square) error between x and $mu_\theta(z)$, therefore $\log p_\theta(x|z)$ is a square reconstruction error between these terms - measuring the

distance between the data points and the $\mu_\theta(z)$. In inverse problems, this SSE term is also known as a data fidelity term. The second term $D_{KL}[q_\phi(z|x)||p_\theta(z)]$ measures the difference between the two density functions, this is intuitively reasonable as $q_\phi(z|x)$ is the learned distribution of latent variables and $p_\theta(z)$ is the prior - hence adopting the task of a regularizer.

On the condition log likelihood loss term is present but KL divergence term is absent: the reconstruction loss encourages the distribution to describe the likelihood (input data) x, ignoring the importance of the prior, allowing for significant deviation from the prior. However on the condition KL divergence term is present but log likelihood loss term is absent: 'cheating' will occur, as the network will learn narrow distributions - close to the prior - with a small enough variance these distributions effectively represent single values (hence producing a standard autoencoder) and unable to handle ambiguous new data points/random input. Finally in the case when both are present: the terms 'trade-off' advantages and disadvantages to balance each other out, the log likelihood reconstruction error ensures the variation of the data is sufficiently accounted for whilst KL-divergence ensures the distribution is adequately close to the prior.

In the special case where the prior follows a unit variance, zero mean, Gaussian distribution $p_\theta(z) \sim \mathcal{N}(0,1)$ and we want $q_\phi(z|x)$ to be described by $\sim \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x))$ (of the same family of distributions) then sampling the distribution of $q_\phi(z|x)$ is effortless - allowing for a closed loop KL-divergence expression. As such, by substituting these parameter values into equation 5 the regularizer term $D_{KL}[q_\phi(z|x)||p_\theta(z)]$ can be vastly simplified to:

$$\begin{aligned} D_{KL}[q_\phi(z|x)||p_\theta(z)] &= D_{KL}[\mathcal{N}(\mu_\phi(x), \Sigma_\phi(x)), \mathcal{N}(0,1)] \\ &= \frac{1}{2}[Tr(\Sigma_\phi(x)) + \mu_\phi(x)^T \mu_\phi(x) - K - \log|\Sigma_\phi(x)|] \end{aligned}$$

Moreover K is the dimension of the Gaussian; the trace function is the sum of the diagonals of the covariance matrix $\Sigma_\phi(x)$ and the determinant of a diagonal matrix $|\Sigma_\phi(x)|$ is the product of its diagonals. The above can thus simplify as:

$$\begin{aligned} &= \frac{1}{2}[\sum_K \Sigma_\phi(x) + \sum_K \mu_\phi^2(x) - \sum_K 1 - \log \prod_K \Sigma_\phi(x)] \\ &= \frac{1}{2}[\sum_K \Sigma_\phi(x) + \sum_K \mu_\phi^2(x) - \sum_K 1 - \sum_K \log \Sigma_\phi(x)] \\ &= \frac{1}{2}\sum_K [\Sigma_\phi(x) + \mu_\phi^2(x) - 1 - \log \Sigma_\phi(x)] \end{aligned}$$

which denotes a simple closed loop expression. It may, however, be more accurate to model $\Sigma_\phi(x)$ as $\exp \Sigma_\phi(x)$ for numerical stability.