# Variational Auto Encoder Latent Space Activity and Visualization

**Jesse Bettencourt**
Department of Mathematics
University of Toronto
Toronto, ON
jessbett@math.toronto.edu

**Matt Craddock**
Department of Computer Science
University of Toronto
Toronto, ON
matt.craddock@mail.utoronto.ca

## Abstract

The abstract paragraph should be indented 1/2 inch (3 picas) on both left and right-hand margins. Use 10 point type, with a vertical spacing of 11 points. The word **Abstract** must be centered, bold, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

## 1 Introduction to Variational Autoencoders

$$\log p(x) \geq \mathbb{E}_{q_\phi(z|x)}[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}] = -\mathcal{L}(x) \tag{1}$$

where

$$\mathcal{L}(x) = D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right) - \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] \tag{2}$$

Variational auto-encoders (VAEs), as defined by Kingma & Welling, 2013 [1], are probabilistic implementations of traditional auto-encoders in which neurons in a latent layer are treated not as fixed values but as probability distributions. This is a major boon to semi-supervised learning (Kingma et al., 2014 [2]) as it allows for non-deterministic generation of data with labels to be trained on. During training, the VAE builds a recognition network represented by the function $q(\mathbf{z}|\mathbf{x};\theta)$ to predict the nodes of a latent space vector $\mathbf{z}$ composed of randomly distributed variables, given a data vector $\mathbf{x}$ and parameters $\theta$, as well as a generative network $p(\mathbf{x}|\mathbf{z};\theta)$ to produce the latter given the former. This is done by maximizing the lower bound on the likelihood function given by:

In this equation the expected value term measures the likelihood of the generative network by measuring the log likelihood of a reconstruction of the data, while the KL divergence term measures the likelihood of the recognition network by comparing the recognized posterior probability to the real posterior. The loss function, $-L(\mathbf{x}^{(i)})$, is minimized by way of gradient descent, which can be made more computationally efficient by the use of the stochastic Adam algorithm, researched by Kingma and Ba, 2015 [3]. In an intuitive sense, the latent space distribution $q(\mathbf{z}|\mathbf{x};\theta)$ is a generic representation of the class of the data vector $\mathbf{x}$. When the distribution of $\mathbf{z}$ is taken to be Gaussian, as it will be in this experiment, the means of the distribution $E_q[\mathbf{z}]$ intuitively represent a prototypical element of the class seen in the label. Based on this idea, Burda, Grosse, & Salakhutdinov, 2016 [4] considered the "activity" of a particular node as:

$$Cov_{\mathbf{x}}\left(\mathbb{E}_{u\ q(u|\mathbf{x})}[u]\right) > \epsilon \tag{3}$$

In this equation, $u$ is a single dimension of the full latent vector $\mathbf{z}$ and $\epsilon$ is a threshold above which the covariance function indicates substantial activity. This measurement, which reflects the variance of the distribution means in a particular dimension, is a measurement of how pronounced the change in

distribution is across different classes in the data, where a higher propensity for change represents a high significance in both predicting and generating (that is, characterizing in the latent space) data of a particular class. The research cited suggests that in general most dimensions of a high dimensional latent space are "pruned out" during training. That is, they are not used to store any significant information about the data, and thus are in a sense wasteful. One of the goals of this experiment is to determine whether this spatial inefficiency in the latent space is simply an inconsequential byproduct of the training process or whether it has a significant impact on the effectiveness of the VAE. A body of research is emerging, such as Johnson, et al. 2016 [5], Burda, Grosse, & Salakhutdinov, 2016 [4], and Snderby et al., 2016 [6] to test various additions and modifications to the original definition of VAEs in order to improve the effectiveness of the networks themselves as well as the process of training them. Snderby et al., 2016 [6] attempted to better utilize the full dimensionality of the latent space by modifying the training process. In the experiment, two features are added to the VAE with the express purpose of preventing nodes with small contributions from being quickly disregarded. The first is batch normalization (proposed by Ioffe & Szegedy, 2015 [7]), wherein the weights of nodes in the deterministic layers of the VAE are normalized by their first moment. This prevents weights of different nodes in the same layer from becoming strongly divergent, with certain nodes contributing a much larger share of information to successive layers than others due to their increased weights. The second is called warmup, in which a scaling parameter is added to the negative of the likelihood function in Equation (1), producing the modified loss function:

$$-\mathcal{L}(\mathbf{x}^{(i)}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x}^{(i)};\theta)}\left[\log p(\mathbf{x}^{(i)}|\mathbf{z};\theta)\right] - \beta D_{KL}\left(q(\mathbf{z}|\mathbf{x}^{(i)};\theta)||p(\mathbf{z})\right) \tag{4}$$

The value of $\beta$ is initialized to zero, and scaled up linearly over the training epochs so that the loss due to the generative network, which is strongly dependent on the state of the latent space, is not a strong factor in early training stages, and thus nodes in the latent space that have little relevance early in the training process are not prevented from having their weights increased later in the process by virtue of their irrelevance. Rather, what would otherwise be a large loss in the generative network is tolerated early on, giving time for more latent space dimensions to become active over different data vectors. The experiment discovered that these two features had a strong effect on the number of active latent space dimensions, thus increasing the spatial efficiency of the stochastic layer. With these examples as precedent, we will proceed first with several attempts to increase the spatial efficiency (equivalently, the number of active dimensions) of the latent space using an array of features and modifications added to the canonical implementation of VAEs by Kingma & Welling, 2013 [1]. We will then conduct an experiment to measure the effect that a change in the value of our activity metric has on the overall performance of the VAE.

## 2 Implementing Variational Autoencoders

In this section we will detail methods for VAE implementation and effect on learning rate.

### 2.1 Network Architecture

Each of the following methods were included into what will we call the 'Vanilla VAE'. The network architecture for the Vanilla VAE and all additional methods is similar to the architecture introduced in the Section 3 example of Auto-Encoding Variational Bayes [4].
The encoder and decoder sub-networks are symmetric, simple, fully-connected neural networks, namely Multi Layer Perceptrons (MLP). Both feature two deterministic, or hidden, layers each with 200 dimensions (or nodes) per layer. A stochastic, or latent, layer with dimensions $n_z$ on top the deterministic layers. In hidden layers the activation is the softplus function.
Practically, the probabilistic encoder network takes data from the input space and encodes a representation into a latent space with dimension $n_z$. In particular, this latent representation, $q_\phi(z \mid x)$ is a Gaussian distribution over the possible latent values of $z$ from which data $x$ could have been generated. The probabilistic decoder network takes a latent representation and produces a distribution $p_\theta(x \mid z)$ over possible data values $x$ generated by $z$.
We implement this network and the following additional methods in Tensorflow. We learn the MLP weights and bias parameters, representing the $\phi$ and $\theta$ distribution parameters, with Adam optimiza-

tion minimizing $\mathcal{L}(x)$ with parameters $\beta_1 = 0.9, \beta_2 = 0.9, \epsilon = 10^{-4}$ with batch size $100$, learning rate $0.001$, trained for $300$ epochs.

### 2.2 Xavier Initialization

All parameters in the MLP were initialized with the Xavier-Glorot method outlined in [2]. Xavier-Glorot initialization is shown to improve learning in deep networks by establishing a reasonable range for initial values. Especially for deep networks there is an initial value trade-off. If the initial weights are too small then the signal will shrink through the layers and and the influence will trend too small to be useless. If the weights are too large then the signal growth through the layers will trend to large to be representative.
Xavier-Glorot initialization addresses this trade-off by sampling initialization weights from a Gaussian distribution with zero mean and variance as a function of the network connections for the node. The variance for weight $w$ of a neuron is given as a function of the number of neurons feeding into it $n_{in}$ and the number of neurons the result feeds to $n_{out}$

$$\text{Var}(w) = \frac{2}{n_{in} + n_{out}}$$

### 2.3 Decoder Distribution

As mentioned previously, the probabilistic decoder network takes a latent representation $z$ and produces a distribution over possible data values, $p_\theta(x \mid z)$. In our initial description of the network architecture we specified that output of the encoder MLP is Gaussian, but we made no specification to the decoder output distribution.
Two choices for decoder distributions were outlined in *Auto-Encoding Variational Bayes*, where the authors suggest that the choice of preferred decoder distribution depends on the type of data [4].

#### 2.3.1 Gaussian Decoder and Encoder Structure

For continuous, real-valued data, Kingma & Welling suggest letting $p_\theta(x \mid z)$ be a multivariate Gaussian distribution. This gives the following structure for the decoder distribution with two hidden deterministic layers $h_1$ and $h_2$:

$$\log p_\theta(x \mid z) = \log \mathcal{N}(x; \mu, \sigma^2 I)$$

where

$$\begin{aligned} \mu &= \text{sigmoid}(W_\mu h_2 + b_\mu) \\ \log \sigma^2 &= \tanh(W_\sigma h_2 + b_\sigma) \\ h_2 &= \text{softplus}(W_2 h_1 + b_2) \\ h_1 &= \text{softplus}(W_1 z + b_1) \end{aligned}$$

Note that the parameters $\{W_1, W_2, W_\mu, W_\sigma, b_1, b_2, b_\mu, b_\sigma\}$ here are the learned parameters of the decoder MLP, and represent the decoder distribution parameter, $\theta$ in $p_\theta(x \mid z)$. Further, since our encoder distribution is always a multivariate Gaussian distribution, we use this structure for the encoder, where the $z$ and $x$ are swapped and the weights and biases represent the encoder distribution parameter, $\phi$ in $q_\phi(z \mid x)$.

#### 2.3.2 Bernoulli Decoder Structure

For binary data, Kingma & Welling suggest letting $p_\theta(x \mid z)$ be a multivariate Bernoulli distribution. This gives the following structure for our decoder distribution with two hidden deterministic layers $h_1$ and $h_2$:

$$\log p_\theta(x \mid z) = \sum_{i=1}^{D} x_i \log y_i + (1 - x_i) \cdot \log(1 - y_i)$$

where

$$\begin{aligned} y &= \text{sigmoid}(W_\mu h_2 + b_\mu) \\ h_2 &= \text{softplus}(W_2 h_1 + b_2) \\ h_1 &= \text{softplus}(W_1 z + b_1) \end{aligned}$$

Again, here the Bernoulli decoder distribution parameter $\theta$ is represented by the learned MLP weights and biases $\theta = \{W_1, W_2, W_\mu, b_1, b_2, b_\mu\}$.

#### 2.3.3 Comparing Decoder Distributions

Our findings support the recommendation by Kingma & Welling that Bernoulli decoder distribution performs better on binary data than a Gaussian decoder. The one-hot MNIST data used to train our VAE is binary, and we expected that the Bernoulli decoder would out-preform the Gaussian decoder. The results of this experiment can be found in Figure

### 2.4 Importance Weighting

In their 2015 publication *Importance Weighted Autoencoders*, Burda, Grosse, and Salakhutdinov observe that Kingma & Welling's $\mathcal{L}(x)$ lower bound on the log-likelihood from Eq.2 makes strong assumptions about the posterior inference leading to overly simplified representations. They propose an improvement called Importance Weighted Auto Encoders (IWAE) which is a generative model using the VAE network architecture, but with a few key improvements to model generalizability.
The critical feature of IWAE is that the encoder network uses multiple importance weighted samples to approximate the posterior, where VAE uses a single sample. This allows IWAE to approximate complex posteriors which are not available under the stronger VAE posterior assumptions.
By considering multiple importance weighted samples we introduce a new lower bound on the log-likelihood which is strictly tighter than Eq.2.
Given $K$ independent samples $\{z_1, \ldots, z_K\}$ from the encoder distribution $z_k \sim q_\phi(z|x)$ we define the new lower bound given by the $K$-sample importance weighting expectation of the log-likelihood:

$$\mathcal{L}_K(x) = \mathbb{E}_{z_1,\ldots,z_K \sim q_\phi(z|x)}[\log \frac{1}{K} \sum_{k=1}^{K} \frac{p_\theta(x, z_k)}{q_\phi(z_k|x)}] \tag{5}$$

The term inside the sum is the normalized importance weights for the joint distribution.
Note in particular that the case $K = 1$ corresponds exactly to Kingma & Welling's $\mathcal{L}(x)$ from Eq.2.
Burda, Grosse, and Salakhutdinov show that this importance weighted lower bound is strictly tighter than the vanilla lower bound. In particular, that $\log p(x) \geq \mathcal{L}_{K+1}(x) > \mathcal{L}_K(x)$
Our implementation of IWAE with this lower bound supports their findings. We observed that with $K = 5$ samples IWAE significantly improves the learned log-likelihood bound during training. The results of this experiment can be found in Figure

### 2.5 Batch Normalization

Batch normalization is a recent method developed to improve stability and convergence speed in deep networks [3]. In the recent paper *How to Train Deep Variational Autoencoders and Probabilistic Ladder Networks* by Sønderby et al. 2016, the authors show that batch normalization is an essential method for learning deep VAEs [**?**]. That is, deep generative models with several latent layers. Though we are only considering a shallow model with a single latent layer, we were interested to see how batch normalization affects training and, as we will discuss later, the latent space

dimensionality.
Batch normalization was developed to improve learning stability during deep network training. As parameters change during learning the layer output distributions change for each hidden layer, requiring later layers to respond to these distribution changes. The problem that batch normalization attempts to address is that changes in early layer output distributions can cause noisy changes to later layers.
Batch normalization addresses this problem by normalizing the inputs of the activation function for each layer so that the inputs across each training batch have a mean of 0 and a variance of 1. However, batch normalization restricts the representation of each layer by assuming this normal distribution. To alleviate some of this restriction, batch normalization introduces learnable parameters to scale the variance of the normal distribution, $\gamma$ and shift the mean, $\beta$. Therefore, we transform each activation function input, $x_i$, with the batch normalization given by:

$$\text{BN}(x_i) = \gamma(\frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}) + \beta$$

where $\mu_B$ and $\sigma$ is the mean and standard deviation of the layer's activation function input across the batch, $\epsilon$ is a small constant to the variance to avoid division by 0, and $\gamma$ and $\beta$ are learnable scale and shift parameters. Note in particular that when learning with batch normalization, the learned shift parameter $\beta$ for each layer replaces the need for bias terms added to that layer's activation inputs.
To train our model we add batch normalization before the activation for all layers in our auto-encoder except the output layers, as described in [**?**].
See the result of training with batch normalization in Figure

### 2.6 Warm Up

Recall that the log-likelihood lower bound in Eq.2 contains a reconstruction term and a variational regularization term. Further, notice that without that variational regularization term the lower bound becomes that for a standard deterministic autoencoder. It has been observed that the variational regularization term causes some latent dimensions to become inactive or 'pruned' during training [5, 1]. In the later sections of this report we consider the activity of the latent dimensions, particularly we are interested in how training maintains or prunes latent dimensions.
Pruning non-informative dimensions later in training could be considered advantageous for automatic relevance determination. However, if latent dimensions are pruned too early in training they will not have a chance to learn informative representations. Once the dimensions become inactive in training, they will not be reactivated. This problem of early latent dimension pruning is particularly troublesome for deep VAEs, because deep latent layers depend on the shallow latent dimensions in the network. If shallow latent dimensions are pruned early, deep latent layers will not be able to learn useful representations [**?**].
To avoid the problem of early pruning due to the variational regularization, we 'warm up' our VAE. Warm-up is achieved by initializing the learning process with a standard deterministic autoencoder, and then linearly introducing the variational regularization. This way the latent dimensions have a chance to learn useful representations as in a deterministic autoencoder before being possibly pruned by variational regularization.
We introduce a warm-up parameter $\beta$ to our objective function which increases linearly from 0 to 1 during the first $N_T$ epochs of training:

$$-\mathcal{L}(x)_T = -\beta D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right) + \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] \tag{6}$$

Again, note that this causes the first epoch to initialize a standard deterministic autoencoder, then linearly introduce the variational behaviour. Further, observe that after $N_T$ epochs the model remains a fully variational autoencoder. This warm-up can also be applied to the lower bound objective of the IWAE identically, by linearly scaling the variational regularization term.
See the result of training with warm-up in Figure

## References

[1] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance Weighted Autoencoders. pages 1–12, 2015.

[2] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9:249–256, 2010.

[3] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv*, 2015.

[4] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. (Ml):1–14, 2013.

[5] David MacKay. Local minima, symmetry-breaking, and model pruning in variational free energy minimization. *Inference Group, Cavendish Laboratory,*, pages 1–10, 2001.