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

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 θ, 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:

$$L(\mathbf{x}^{(i)}) = D_{KL}\left(q(\mathbf{z}|\mathbf{x}^{(i)};\theta)||p(\mathbf{z})\right) - \mathbb{E}_{q(\mathbf{z}|\mathbf{x}^{(i)};\theta)}\left[\log p(\mathbf{x}^{(i)}|\mathbf{z};\theta)\right] \tag{1}$$

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:

$$Cov_{\mathbf{x}}\left(\mathbb{E}_{u\ q(u|\mathbf{x})}[u]\right) > \epsilon \tag{2}$$

In this equation, u is a single dimension of the full latent vector $\mathbf{z}$ and ϵ 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:

$$-L(\mathbf{x}^{(i)}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x}^{(i)};\theta)}\left[\log p(\mathbf{x}^{(i)}|\mathbf{z};\theta)\right] - \beta D_{KL}\left(q(\mathbf{z}|\mathbf{x}^{(i)};\theta)||p(\mathbf{z})\right) \tag{3}$$

The value of β 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 [1].
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 all 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 ϕ and θ distribution parameters, with Adam optimization 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 [**?**]. 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 [1].

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

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

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, θ in $p_\theta(x \mid z)$. Further, since the 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, ϕ 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.

References

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