Shortened introduction to variational autoencoders (VAE)

Or, how to understand a VAE implementation

Juha Jeronen

December 23, 2022

Version 0.3

Contents

1 Introduction

The aim of this document is to condense the main points from the tutorial paper by the inventors of VAE, Kingma and Welling (2019), in order to facilitate understanding VAE implementations; especially the loss computation, and how to interpret the outputs of the neural network components of the system.

It should be kept in mind that essentially, a neural network just produces arbitrary numbers. Intuitively, the "magic" is in how we use those numbers (i.e. what we declare them to represent), how we set up our objective function for training the network, and how the optimization process (training) drives those numbers to actually become the thing we arbitrarily declared them to represent.

We cite other studies where they particularly add value. We add our own notes for clarification where we feel it is necessary. We assume familiarity with classical (deterministic) mathematical modeling and numerics, but no background in stochastics.

2 Problem setup and the proposed solution

Consider a situation where we have acquired a set of N distinct measurements (observations) $\mathbf{x}$, taken from the same, unchanging system. Beside physical modeling from first principles, there is another fundamental approach for modeling for understanding the behavior of the system, providing a different kind of insight and different possibilities for numerical computation.

This other approach is statistical. The critical insight is that repeated observations of an unchanging system can be thought of as samples drawn from an unknown probability distribution implicitly defined by the system.

To characterize the system, we would like to construct a statistical model that approximates the unknown distribution. Let $\mathcal{D}$ be a finite dataset, consisting of N independently and identically distributed (i.i.d.) data points $\mathbf{x}$, which represent observations of our system. We now ask: given some model θ, what is the probability that drawing N i.i.d. samples from that model produces the dataset $\mathcal{D}$?

Taking a bayesian viewpoint, this probability represents our degree of belief in the model θ. Informally, models that yield a higher probability for the observed dataset are better approximations of the true, unknown distribution.

I.i.d. probabilities combine by multiplication, so the probability of the dataset $\mathcal{D}$ under the model θ is

$$p_\theta(\mathcal{D}) = \prod_{\mathbf{x}\in\mathcal{D}} p_\theta(\mathbf{x}) \tag{1}$$

In practice, for numerical stability it is better to work with log-probabilities, so we take the logarithm:

$$\log p_\theta(\mathcal{D}) = \log \prod_{\mathbf{x}\in\mathcal{D}} p_\theta(\mathbf{x}) = \sum_{\mathbf{x}\in\mathcal{D}} \log p_\theta(\mathbf{x}) \tag{2}$$

In other words, the log-probability of the dataset $\mathcal{D}$ under the model θ is the sum of log-probabilities of the individual data points $\mathbf{x}$ under that model. By maximizing this log-probability with respect to θ (over some considered class of models, which we supply), we can find the model — among those considered — that is the most likely. This is known as the **maximum log-likelihood** (ML) criterion.

Maximum log-likelihood is a simple stochastic model selection criterion. Kingma and Welling (2019) point out that more advanced bayesian criteria include *maximum a posteriori* (MAP) estimation, and inference of a full approximate posterior distribution over the parameters. We will use ML for its simplicity.

The quantity $p_\theta(\mathbf{x})$, **when interpreted as a function of** θ, is the **likelihood** (of the data point $\mathbf{x}$, given the model θ), also known as the **model evidence**. In other words, $p_\theta(\mathbf{x})$ is the probability mass assigned to data point $\mathbf{x}$ by the model θ. In yet other words, it measures the strength of evidence that the data point $\mathbf{x}$ provides for the model θ.

A likelihood is **not** a probability density function, because in general,

$$\int_{\theta \in \Theta} p_\theta(\mathbf{x}) \, \mathrm{d}\theta \neq 1 \tag{3}$$

where Θ is the set of all considered models (which can be continuous if we are working with a family of parameterized models, hence the integral). In other words, for θ varying and $\mathbf{x}$ fixed, $p_\theta(\mathbf{x})$ is **not** a probability density function. Hence the different name *likelihood*.

But for any given model (i.e. fixed θ),

$$\int_{\mathbf{x} \in \mathbf{X}} p_\theta(\mathbf{x}) \, \mathrm{d}\mathbf{x} = 1 \tag{4}$$

where $\mathbf{X}$ is the set of all possible observations. In other words, for θ fixed and $\mathbf{x}$ varying, $p_\theta(\mathbf{x})$ **is** a probability density function — it is the probability density of the random variable $\mathbf{x}$ under the model θ.

The likelihood is a slice, *cutting across different models*, of the probability density assigned by each of these models *to a fixed data point* $\mathbf{x}$. In models with latent (unobserved) variables, $p_\theta(\mathbf{x})$ is termed the **marginal likelihood**, because the latent variables $\mathbf{z}$ have been marginalized out (i.e. integrated or summed over, as appropriate).

Some machine-learning models with **latent variables** are especially powerful for **manifold learning**, i.e. the automatic extraction of structure from given data. We will concentrate on a particular subclass of these models: the **variational autoencoder** (VAE).

We denote the latent (unobserved) variables by $\mathbf{z}$. It may help intuition to think of $\mathbf{z}$ as a set of **features** or **explanatory factors** that describe the data $\mathbf{x}$. For example, in a photograph dataset, such features could include the viewing angle and the illumination conditions. These variables are not explicitly provided in the dataset (hence *unobserved* or *latent*), but if discovered, the data can be explained in terms of them.

In autoencoders, the latent space is chosen to have a much lower dimension than the data space. The idea is that by bottlenecking the reconstruction process through a low-dimensional space, then in order for the model to be able to produce a faithful reconstruction of the input data $\mathbf{x}$, it is forced to make the latent variables $\mathbf{z}$ to represent higher-level features that explain the observed variation in the data. Informally, this is especially true if the coded representation can be made regular (in some appropriate sense; continuity is necessary but not sufficient) in terms of small perturbations to the data. This is where the variational autoencoder comes in.

Furthermore, we will impose the latent space to have a simple structure. The premise is that in any real-world application, we can expect the features to follow a simple distribution — provided that we can discover those features. The problem then reduces to learning the (highly nonlinear) mapping between the data space and the latent space — in other words, discovering how to map the seemingly complex data onto features that follow a simple distribution.[1] Thus the optimization of a VAE can also be viewed as a kind of nonlinear PCA (principal component analysis), but with no orthogonality constraint.

An important practical issue is that in realistic applications, the true posterior of the latent variables conditioned on data, $p_\theta(\mathbf{z}|\mathbf{x})$, is intractable to compute, as is the marginal likelihood of the data, $p_\theta(\mathbf{x})$. The now-famous paper by Kingma and Welling (2013) provides a variational method to approximate $p_\theta(\mathbf{z}|\mathbf{x})$, and to choose the optimal model from a considered class. We will now look at the details of this method.

[1]Note that in general, the choice of the features is not unique.

3 The evidence lower bound (ELBO)

The **evidence lower bound**, commonly abbreviated as **ELBO**, is defined for a data point $\mathbf{x}$ as

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) := \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \tag{5}$$

where θ are the parameters of the decoder (a.k.a. generative model, or observation model), and ϕ are the parameters of the encoder (a.k.a. inference model). The encoder parameters ϕ are also known as the **variational parameters** for a reason that will become apparent shortly. Note the expectation over the latent variables $\mathbf{z}$, drawn from an auxiliary distribution $q_\phi(\mathbf{z}|\mathbf{x})$.

Equation (5) is sometimes called the **joint-contrastive** expression of the ELBO.

What is the motivation behind the ELBO? Following Kingma and Welling (2019), consider

$$\begin{aligned}
\mathcal{L}_{\theta,\phi}(\mathbf{x}) &\equiv \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\big] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log\big(p_\theta(\mathbf{x})p_\theta(\mathbf{z}|\mathbf{x})\big) - \log q_\phi(\mathbf{z}|\mathbf{x})\Big] \qquad \text{(rewrite joint probability)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x}) + \log p_\theta(\mathbf{z}|\mathbf{x}) - \log q_\phi(\mathbf{z}|\mathbf{x})\big] \qquad \text{(logarithm arithmetic)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x}) - \big(\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p_\theta(\mathbf{z}|\mathbf{x})\big)\Big] \qquad \text{(regroup)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}) - \log\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \qquad \text{(logarithm arithmetic)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\log p_\theta(\mathbf{x}) - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\log\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} \qquad \text{(linearity of expectation)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \qquad \text{(definition of Kullback–Leibler divergence)} \\
&\equiv \log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \qquad \text{(evaluate the expectation in the first term)}
\end{aligned} \tag{6}$$

so the ELBO is the marginal log-likelihood of the model θ at the data point $\mathbf{x}$, namely $\log p_\theta(\mathbf{x})$, minus the KL divergence of $q_\phi(\mathbf{z}|\mathbf{x})$ from the (intractable) true posterior distribution $p_\phi(\mathbf{z}|\mathbf{x})$.

The ELBO can be computed for any probability density $q_\phi(\mathbf{z}|\mathbf{x})$, and it always gives a guaranteed lower bound for $\log p_\theta(\mathbf{x})$, so it yields a variational method to approximate the otherwise intractable marginal log-likelihood $\log p_\theta(\mathbf{x})$. The tightness of the bound depends on the chosen $q_\phi(\mathbf{z}|\mathbf{x})$. As always with variational techniques, a completely inappropriate $q_\phi(\mathbf{z}|\mathbf{x})$ may give a correct but useless lower bound of $-\infty$, so some care must be taken in choosing $q_\phi(\mathbf{z}|\mathbf{x})$.

Consider now what happens if we maximize the ELBO by varying the distribution $q_\phi(\mathbf{z}|\mathbf{x})$, while keeping everything else fixed. Since $D_{KL} \geq 0$, the maximum with respect to the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is reached when $q_\phi(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{z}|\mathbf{x})$. The marginal log-likelihood term $\log p_\theta(\mathbf{x})$ does not depend on $q_\phi(\mathbf{z}|\mathbf{x})$, so in this maximization, it is a constant.

Thus, if we can find a distribution $q_\phi(\mathbf{z}|\mathbf{x})$ that approximates the intractable true posterior $p_\theta(\mathbf{z}|\mathbf{x})$, the ELBO will approximate the marginal log-likelihood $\log p_\theta(\mathbf{x})$, which is our maximization objective for model selection via the ML criterion. Thus we recognize the auxiliary distribution $q_\phi(\mathbf{z}|\mathbf{x})$ as an **approximate posterior**.

In practice, we will represent $q_\phi(\mathbf{z}|\mathbf{x})$ using a parametric, pre-chosen family of distributions, so in general, it will not be able to perfectly approximate an arbitrary $p_\theta(\mathbf{z}|\mathbf{x})$. The bound will not be tight; but if we choose the family $q_\phi(\mathbf{z}|\mathbf{x})$ to be sufficiently flexible, the bound will however be useful for practical purposes.

Also, in practice we maximize with respect to both θ and ϕ concurrently. This will both approximately maximize $\log p_\theta(\mathbf{x})$, and improve the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$, making it closer to $p_\theta(\mathbf{z}|\mathbf{x})$ within the constraints of the chosen family of distributions $q_\phi(\mathbf{z}|\mathbf{x})$, thus making the bound tighter.

Summing over the ELBOs of all data points $\mathbf{x} \in \mathcal{D}$, we obtain the total ELBO of the model (θ, ϕ). The **ELBO loss** (a.k.a. **VAE loss**) is the negative of the total ELBO; this is our minimization objective for choosing the best model (out of the class considered).

Importantly, the ELBO is computable. We can rewrite it as

$$
\begin{aligned}
\mathcal{L}_{\theta,\phi}(\mathbf{x}) &\equiv \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log\big(p_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z})\big) - \log q_\phi(\mathbf{z}|\mathbf{x})\Big] \qquad \text{(rewrite joint probability the other way)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\underbrace{\log p_\theta(\mathbf{z})}_{\text{latent prior}} + \underbrace{\log p_\theta(\mathbf{x}|\mathbf{z})}_{\text{observation model}} - \underbrace{\log q_\phi(\mathbf{z}|\mathbf{x})}_{\text{approximate posterior}}\Big] \qquad \text{(logarithm arithmetic)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x}|\mathbf{z}) - \big(\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p_\theta(\mathbf{z})\big)\Big] \qquad \text{(regroup)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) \qquad \text{(similarly as before)} \qquad (7)
\end{aligned}
$$

The third (annotated) line is how VAE implementations commonly implement the ELBO, using a single-sample Monte Carlo (MC) estimate of this expectation, discussed below. Importantly, the expectation is taken over $\mathbf{z}$ drawn from the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$, which we have available for sampling. See e.g. Kingma and Welling (2019, algorithm 2).

The conditional distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is called the **inference model** (i.e. or **approximate posterior**, or **encoder**); the intuition is that it attempts to infer features $\mathbf{z}$ from the given data point $\mathbf{x}$.

The conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$ is called the **observation model** (i.e. **generative model**, or **decoder**); it attempts to explain what observations $\mathbf{x}$ could have generated the code point $\mathbf{z}$. Or interpreted differently, given a random code point $\mathbf{z}$, it generates new data that is statistically similar to the members of the dataset $\mathcal{D}$.

The distribution $p_\theta(\mathbf{z})$ is termed the **latent prior**, or just the **prior**. This is because it is not conditioned on any observations. In other words, before looking at any data, $p_\theta(\mathbf{z})$ is the distribution we believe the code points $\mathbf{z}$ should follow.

The last line of (7) is sometimes called the **prior-contrastive** expression of the ELBO. It offers an alternative interpretation: we can also view the ELBO as the expectation — under drawing $\mathbf{z}$ from the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ — of the log-likelihood of the observation model $p_\theta(\mathbf{x}|\mathbf{z})$ at the data point $\mathbf{x}$ (the *reconstruction likelihood*); minus the KL divergence of the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ *from the latent prior* $p_\theta(\mathbf{z})$.

So maximizing the ELBO with respect to $q_\phi(\mathbf{z}|\mathbf{x})$ also has the regularizing effect of pushing the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ closer to the latent prior $p_\theta(\mathbf{z})$, leading to a regularizing effect on the latent representation. (This argument is not fully rigorous, because in the prior-contrastive expression, the choice of $q_\phi(\mathbf{z}|\mathbf{x})$ affects also the first term.)

Some VAE implementations insert a weighting hyperparameter on the D_{KL} term, to tune the relative contributions of the reconstruction quality (measured by the term $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]$) and the latent regularization (which arises due to the D_{KL} term). We recognize this as the classical weighting method from multiobjective optimization. Lin et al. (2019) point out that the terms indeed represent conflicting objectives.

Lin et al. (2019) also provide an interesting perspective here, on how to automatically find an optimal balance between reconstruction quality and regularization. We will return to this point when we discuss the choice of the observation model $p_\theta(\mathbf{x}|\mathbf{z})$ below.

4 Making the ELBO computable

Following Kingma and Welling (2013), we reparameterize the latent variable $\mathbf{z}$, which follows the distribution $q_\phi(\mathbf{z}|\mathbf{x})$, as a deterministic transformation of a new random variable $\boldsymbol{\varepsilon}$:

$$\mathbf{z} := \mathbf{g}(\boldsymbol{\varepsilon}, \phi, \mathbf{x}) \tag{8}$$

In practice, the transformation is often defined in two parts. A typical $\mathbf{g}$ consists of a simple explicit function, with parameters (that depend on the data $\mathbf{x}$ in a highly nonlinear manner) that are computed by a neural network. We will see an example when we discuss the choice of the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$.

The variable $\boldsymbol{\varepsilon}$ is termed the **noise variable**. We choose the noise to follow a simple, non-parametric distribution, such as a spherical Gaussian:

$$p(\boldsymbol{\varepsilon}) := \mathcal{N}(\boldsymbol{\varepsilon}, 0, \mathbf{1}) \tag{9}$$

This is known as *the reparameterization trick*, and is a critically important detail for successful implementation of a VAE. The stochasticity of $\mathbf{z}$ is now isolated into the new non-parametric random variable $\boldsymbol{\varepsilon}$. The original variable $\mathbf{z}$ has become differentiable with respect to the parameters of our transformation $\mathbf{g}$. When used in neural networks (technically, directed graphical models), this allows backpropagation to work across graph nodes involving $\mathbf{z}$, while maintaining the stochastic nature of $\mathbf{z}$. See Kingma and Welling (2019, figure 2.3).

Using the reparameterization, we change variables in the expectation:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] = \mathbb{E}_{p(\boldsymbol{\varepsilon})}[f(\mathbf{z})] \tag{10}$$

where f is any differentiable function. Note the integrand on the right-hand side is still written in terms of the original variable $\mathbf{z}$, so no jacobian determinant factor appears.

This change of variable brings the advantage that now the expectation itself no longer depends on the parameters ϕ, so the gradient ∇_ϕ and the expectation $\mathbb{E}_{p(\boldsymbol{\varepsilon})}$ commute. This in turn allows evaluating a single-sample Monte Carlo estimate of the gradient as

$$\begin{aligned}
\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] &= \nabla_\phi \mathbb{E}_{p(\boldsymbol{\varepsilon})}[f(\mathbf{z})] \\
&= \mathbb{E}_{p(\boldsymbol{\varepsilon})}[\nabla_\phi f(\mathbf{z})] \\
&\simeq \nabla_\phi f(\mathbf{z})
\end{aligned} \tag{11}$$

where the symbol $\simeq$ means that one side (here the right-hand side) is an unbiased estimator of the other. On the last line, $\mathbf{z}$ is evaluated by drawing **one** noise sample $\boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon})$, and applying the current iterate of the transformation $\mathbf{g}$ — so that in effect, $\mathbf{z}$ becomes drawn from the current approximate posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$, as we indeed need for approximating the expectation in terms of this distribution. These tricks make the estimate computable, as well as allow us to use stochastic gradient descent (SGD) on the ELBO.

Similarly, a single-sample Monte Carlo estimate of the ELBO itself can be written as

$$\begin{aligned}
\mathcal{L}_{\theta,\phi}(\mathbf{x}) &\equiv \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
&= \mathbb{E}_{p(\boldsymbol{\varepsilon})}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
&\simeq \log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})
\end{aligned} \tag{12}$$

where we evaluate $\mathbf{z}$ by drawing one noise sample $\boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon})$, and applying the current iterate of the transformation $\mathbf{g}$.

Thus to compute the ELBO, given a pair $(\mathbf{x}, \mathbf{z})$, we actually only need to evaluate the log-densities $\log p_\theta(\mathbf{x}, \mathbf{z})$ and $\log q_\phi(\mathbf{z}|\mathbf{x})$. The joint log-density is usually further split into two terms, as on the annotated line of equation (7). The gradient of the ELBO with respect to θ and ϕ is then obtained by backpropagation (backward mode automatic differentiation).

Keep in mind that the expectation that appears in the ELBO is $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}$, so when evaluating the single-sample Monte Carlo estimate of the ELBO, we must use a $\mathbf{z}$ sampled from $q_\phi(\mathbf{z}|\mathbf{x})$ in all terms — especially, also in the log-prior term $p_\theta(\mathbf{z})$. That is, for this term, we evaluate the log-density that *the prior distribution* $p_\theta(\mathbf{z})$ assigns to the $\mathbf{z}$ sample drawn *from the approximate posterior* $q_\phi(\mathbf{z}|\mathbf{x})$.

5 The distributions $p_\theta(\mathbf{z})$, $p_\theta(\mathbf{x}|\mathbf{z})$, and $q_\phi(\mathbf{z}|\mathbf{x})$

To completely specify a VAE, we need to choose three distributions: a latent prior $p_\theta(\mathbf{z})$, a class of observation models $p_\theta(\mathbf{x}|\mathbf{z})$, and a class of approximate posteriors $q_\phi(\mathbf{z}|\mathbf{x})$.

The **latent prior** $p_\theta(\mathbf{z})$ can be taken to be some simple distribution with no parameters; this then imposes a soft restriction on the codes the VAE can produce. (The subscript θ, indicating the decoder parameters, then becomes superfluous, so we could write just $p(\mathbf{z})$.)

The classic VAE by Kingma and Welling (2013) uses a spherical Gaussian latent space, which is still perhaps the most popular latent space for VAEs:

$$p_\theta(\mathbf{z}) := \mathcal{N}(\mathbf{z}; 0, \mathbf{1}) \tag{13}$$

Note this is distinct from the spherical Gaussian we used for the noise variable $\boldsymbol{\varepsilon}$. Here we specify the prior for the code points $\mathbf{z}$, whereas $p(\boldsymbol{\varepsilon})$ is just an auxiliary distribution used for reparameterizing the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ to make it usable with optimization algorithms that use backpropagation.

The **observation model** in the classic VAE is a factorized Bernoulli:

$$\log p(\mathbf{x}|\mathbf{z}) := \sum_{j=1}^{D} \log p(x_j|\mathbf{z}) \tag{14}$$

where

$$p(x_j|\mathbf{z}) := \text{Bernoulli}(x_j; p_j) \equiv p_j^{x_j} + (1 - p_j)^{1-x_j} \tag{15}$$

which yields the log-density

$$\log p(x_j|\mathbf{z}) = x_j \log p_j + (1 - x_j) \log(1 - p_j) \tag{16}$$

which is the form used in equation (1.18) in Kingma and Welling (2019). The latent variable $\mathbf{z}$ is mapped to the Bernoulli parameter vector $\mathbf{p}$ by training a **decoder neural network** (NN), parameterized by the decoder parameters θ:

$$\mathbf{p} := \text{NN}_\theta(\mathbf{z}) \tag{17}$$

The neural network is where "the magic happens": it is the part of the VAE that actually establishes the nonlinear mapping from the code space to the data space.

Many papers and VAE tutorials are based on this classic architecture. Note that the Bernoulli distribution models the data as binary values $\{0, 1\}$, instead of a continuous variable in $[0, 1]$. The focus has shifted after Loaiza-Ganem and Cunningham (2019) pointed out that for generic continuous data, this is simply wrong, and that a much better VAE reconstruction (sharper images for the same dimension of latent space) can be obtained by modifying the Bernoulli distribution into its continuous analogue. In practice, this introduces a scaling factor so that the probability density integrates to 1 when x is continuous, and changes the computation of the mean (it is no longer simply the Bernoulli parameter p_j).

Other observation models are also possible. The variance of the observation model (decoded data) is classically taken to be fixed — so the distribution of the observation model is only parameterized by a learnable mean — but also models with learnable observation variance have been suggested. Here particularly interesting is the factorized Gaussian observation model, with a global variance parameter σ^2, proposed by Lin et al. (2019):

$$\log p(\mathbf{x}|\mathbf{z}) := \sum_{j=1}^{D} \log p_\theta(x_j|\mathbf{z}) \tag{18}$$

where

$$p_\theta(x_j|\mathbf{z}) := \mathcal{N}(x_j; \mu_j, \sigma^2) \equiv \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{[x_j - \mu_j]^2}{2\sigma^2}\right) \tag{19}$$

The parameter σ^2 reflects the global noise properties of the data. The latent variable $\mathbf{z}$ is mapped to the data-space parameter vector $\boldsymbol{\mu} = \boldsymbol{\mu}_\theta(\mathbf{z})$, representing the mean of each component in the data space (as a highly nonlinear function of $\mathbf{z}$), by training a decoder neural network, parameterized by the decoder parameters θ:

$$\boldsymbol{\mu}_\theta(\mathbf{z}) := \mathrm{NN}_\theta(\mathbf{z}) \tag{20}$$

The idea of automatic regularization tuning is to learn the value of σ (starting from an initial value of 1) as part of the ELBO optimization. The authors also provide a closed-form solution for the optimal σ_* when θ and ϕ are fixed. The final learned σ represents the optimal amount of noise that we must assume the data to have to make our best-fit model fit the data; it thus also measures the quality of the fit.

The variance parameter can also be made local (like the mean $\boldsymbol{\mu}$ already is), which yields even better results as well as a local uncertainty indicator. However, then some care needs to be taken in designing the training algorithm; for details, see the original study by Lin et al. (2019).

Finally, we need to specify the family of distributions we would like to use as the **approximate posterior** $q_\phi(\mathbf{z}|\mathbf{x})$. Note that this describes the distribution of code points in the latent space.

Keep in mind that in the posterior, we have reparameterized $\mathbf{z} = \mathbf{g}(\boldsymbol{\varepsilon}, \phi, \mathbf{x})$, so we may start from the noise log-density $\log p(\boldsymbol{\varepsilon})$, and transform that via $\mathbf{g}$. Recall that in equation (9), we chose the noise variable $\boldsymbol{\varepsilon}$ to follow the unit spherical Gaussian distribution,

$$p(\boldsymbol{\varepsilon}) \equiv \mathcal{N}(\boldsymbol{\varepsilon}, 0, \mathbf{1})$$

whereas in the ELBO evaluation, the latent variable $\mathbf{z}$ follows the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$.

Following Kingma and Welling (2019), for a given transformation $\mathbf{z} = \mathbf{g}(\boldsymbol{\varepsilon}, \phi, \mathbf{x})$, the log-densities $\log p(\boldsymbol{\varepsilon})$ and $\log q_\phi(\mathbf{z}|\mathbf{x})$ are related by the change of variable in a probability density (which is a relative of the change of variable in an integral):

$$\log q_\phi(\mathbf{z}|\mathbf{x}) = \log p(\boldsymbol{\varepsilon}) - \log d_\phi(\mathbf{x}, \boldsymbol{\varepsilon}) \tag{21}$$

where

$$\log d_\phi(\mathbf{x}, \boldsymbol{\varepsilon}) = \log \left| \det \frac{\partial \mathbf{z}}{\partial \boldsymbol{\varepsilon}} \right| \tag{22}$$

and $\partial\mathbf{z}/\partial\boldsymbol{\varepsilon}$ is the jacobian matrix of the transformation $\mathbf{g}$:

$$\left[\frac{\partial \mathbf{z}}{\partial \boldsymbol{\varepsilon}}\right]_{ik} \equiv \frac{\partial z_i}{\partial \varepsilon_k} \tag{23}$$

Note that in equation (21), we have rewritten the right-hand side in terms of $\boldsymbol{\varepsilon}$ instead of $\mathbf{z}$, hence a jacobian determinant term appears. We use the notation $d_\phi(\mathbf{x}, \boldsymbol{\varepsilon})$ to emphasize that this quantity depends not only on $\boldsymbol{\varepsilon}$, but also on the data $\mathbf{x}$ and the encoder parameters ϕ.

Now it remains to choose a suitable transformation $\mathbf{g}$. Doing so will specify the family of approximate posteriors available for use by the VAE. Informally, it is advantageous to choose something flexible (to be able to reasonably approximate an arbitrary unknown true posterior $p_\theta(\mathbf{z}|\mathbf{x})$), yet having a simple log-determinant $\log d_\phi(\mathbf{x}, \boldsymbol{\varepsilon})$. The latter consideration is for simplicity and computational efficiency, since during optimization, we will need to evaluate the single-datapoint ELBO estimate in the innermost loop.

The classic VAE uses, for simplicity, a factorized Gaussian approximate posterior:

$$q_\phi(\mathbf{z}|\mathbf{x}) := \prod_i q_\phi(z_i|\mathbf{x}) := \prod_i \mathcal{N}(z_i; \mu_i, \sigma_i^2) \tag{24}$$

How does this fit into the transformation framework? We can parameterize the factorized Gaussian by its mean and log-variance vectors $\boldsymbol{\mu}$ and $\log \boldsymbol{\sigma}$. The data points $\mathbf{x}$ are mapped to the parameters of the approximate posterior by training an **encoder neural network** (NN), parameterized by the encoder parameters ϕ:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) := \mathrm{NN}_\phi(\mathbf{x}) \tag{25}$$

Now we can write the transformation (this is the simple explicit function part)

$$\mathbf{z} := \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon} \tag{26}$$

where $\odot$ denotes the elementwise product. In other words, we take the noise distribution $\mathcal{N}(\boldsymbol{\varepsilon}; 0, \mathbf{1})$, and shift and scale each vector component j such that the resulting vector $\mathbf{z}$ follows a factorized Gaussian distribution. When evaluating the ELBO during neural network optimization, we draw a noise sample $\boldsymbol{\varepsilon}$, and then apply (26) to obtain a $\mathbf{z}$ drawn from the current iterate of $q_\phi(\mathbf{z}|\mathbf{x})$.

Equations (25) and (26) together form a concrete example of the transformation $\mathbf{z} = \mathbf{g}(\boldsymbol{\varepsilon}, \phi, \mathbf{x})$. In (26), the dependences of $\mathbf{z}$ on ϕ and $\mathbf{x}$ have been absorbed into $\boldsymbol{\mu} = \boldsymbol{\mu}_\phi(\mathbf{x})$ and $\boldsymbol{\sigma} = \boldsymbol{\sigma}_\phi(\mathbf{x})$, via the neural network $\mathrm{NN}_\phi(\mathbf{x})$.

Again, the neural network is where "the magic happens": it is the part of the VAE that actually establishes the nonlinear mapping from the data space to the code space.

With the choice (26), the jacobian of the transformation $\mathbf{g}$ is just

$$\frac{\partial \mathbf{z}}{\partial \boldsymbol{\varepsilon}} = \mathrm{diag}(\boldsymbol{\sigma}) \equiv \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_N \end{bmatrix} \tag{27}$$

so the log-determinant is found to be

$$\log d_\phi(\mathbf{x}, \boldsymbol{\varepsilon}) = \log \left| \det \frac{\partial \mathbf{z}}{\partial \boldsymbol{\varepsilon}} \right| = \sum_i \log \sigma_i \tag{28}$$

where we have dropped the absolute value since the standard deviations σ_i are nonnegative. The approximate posterior log-density becomes

$$\log q_\phi(\mathbf{z}|\mathbf{x}) = \log p(\boldsymbol{\varepsilon}) - \log d_\phi(\mathbf{x}, \boldsymbol{\varepsilon}) = \sum_i \left[\log \mathcal{N}(\varepsilon_i; 0, 1) - \log \sigma_i \right] \tag{29}$$

This is then evaluated using the same single noise sample $\boldsymbol{\varepsilon}$ as above for evaluating $\mathbf{z}$.

For more details, see Kingma and Welling (2019, sec. 2.4 and 2.5, and algorithm 2).

6 ELBO and Bayes' theorem

Starting from the tautology

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}, \mathbf{z}) \tag{30}$$

we rewrite both sides, splitting the joint probability both ways:

$$p_\theta(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x}) = p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) \tag{31}$$

Rearranging yields **Bayes' theorem** in its standard form — i.e. how to update our beliefs on the latent $\mathbf{z}$ when new data $\mathbf{x}$ arrives:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} \tag{32}$$

However, in evaluating the ELBO loss, this is used in the rearranged form

$$p_\theta(\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})} \tag{33}$$

to evaluate the marginal likelihood of the model θ, given the other three distributions.[2] Now, let us take the logarithm of (33), and then marginalize out $\mathbf{z}$ on the right-hand side by taking an expectation over $\mathbf{z} \sim p_\theta(\mathbf{z}|\mathbf{x})$ on both sides. We have

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{z}) + \log p_\theta(\mathbf{x}|\mathbf{z}) - \log p_\theta(\mathbf{z}|\mathbf{x}) \right] \tag{34}$$

Compare to the annotated line of equation (7), repeated here for convenience:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{z}) + \log p_\theta(\mathbf{x}|\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \right] \tag{35}$$

Since the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is intractable, we approximate it variationally by $q_\phi(\mathbf{z}|\mathbf{x})$. Note also the expectation is now taken over $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$, no longer $\mathbf{z} \sim p_\theta(\mathbf{z}|\mathbf{x})$. As was shown by algebraic manipulation (equation (6)), $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ is a variational lower bound for $p_\theta(\mathbf{x})$, so maximizing this expression over $q_\phi(\mathbf{z}|\mathbf{x})$ yields the maximal lower bound for $p_\theta(\mathbf{x})$ within the chosen family of posteriors $q_\phi(\mathbf{z}|\mathbf{x})$.

We emphasize that just like when choosing a finite element basis, we must choose a family $q_\phi(\mathbf{z}|\mathbf{x})$ that can reasonably represent an approximation of the unknown $p_\theta(\mathbf{z}|\mathbf{x})$; otherwise the obtained lower bound may be useless. A classical numericist observes here that the ELBO optimality guarantee is similar to the residual L^2 orthogonality guarantee of Galerkin methods, in that the result can only be as good as the chosen basis is at representing a useful approximation of the unknown quantity.

[2]This form looks slightly suspect in that the left-hand side depends only on $\mathbf{x}$, whereas the right-hand side depends on both $\mathbf{x}$ and $\mathbf{z}$. This suggests that the $\mathbf{z}$ dependences on the right-hand side must cancel out; but to do this rigorously, it is better to start from (31), apply the logarithm and the expectation over $\mathbf{z}$, and rearrange only after that has been done.

7 ELBO loss evaluation

A VAE is trained by minimizing the total ELBO loss (i.e. maximizing the total ELBO) over the dataset $\mathcal{D}$. We now look at the inner loop that evaluates the ELBO loss for a single data point $\mathbf{x} \in \mathcal{D}$.

To evaluate the ELBO, we use the annotated line of equation (7). One final detail is that since the ELBO loss evaluation must run in the inner loop, we cannot use any expensive methods to evaluate the expectation over $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. So we do this approximately, with a single-sample Monte Carlo estimate; we have everything we need to draw a sample from the current iterate of the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$.

The algorithm is as follows:

- **Input**: data point $\mathbf{x}$.
- **Encode**: compute $\mathrm{NN}_\phi(\mathbf{x})$, obtain parameters for reparameterization transformation $\mathbf{g}$.
- Draw **one** noise sample ε (e.g. from distribution (9)).
- Plugging the parameters into $\mathbf{g}$, transform the noise sample to obtain one $\mathbf{z}$ drawn from $q_\phi(\mathbf{z}|\mathbf{x})$.
- Using this $\mathbf{z}$ sample, compute a single-sample MC estimate of the ELBO:
  - Evaluate approximate posterior log-density $\log q_\phi(\mathbf{z}|\mathbf{x})$ at this $\mathbf{z}$ (e.g. equation (29)).
  - Evaluate latent prior log-density $\log p_\theta(\mathbf{z})$ at this $\mathbf{z}$ (e.g. based on equation (13)).
  - **Decode**: compute $\mathrm{NN}_\theta(\mathbf{z})$, obtain parameters for observation model.
  - Using these parameters, evaluate observation log-density $\log p_\theta(\mathbf{x}|\mathbf{z})$ at the *input* $\mathbf{x}$.
- Combine the three log-densities to obtain the ELBO estimate (see equation (7)).
- **Output**: ELBO loss (negative of the ELBO).

We evaluate $\log p_\theta(\mathbf{x}|\mathbf{z})$ at the *input* $\mathbf{x}$ for two reasons: our $\mathbf{z}$ drawn from $q_\phi(\mathbf{z}|\mathbf{x})$ was conditioned on this value of $\mathbf{x}$, and this is the $\mathbf{x}$ value for which we are computing the single-datapoint estimate of $\mathcal{L}_{\theta,\phi}(\mathbf{x})$.

The encode-decode cycle used in autoencoder (AE) training is still there, but it is now somewhat hidden under the various details. In a VAE, instead of directly producing a point in the other space (latent or data), the neural network parts produce parameters for a parametric distribution. Also in a VAE, the optimization objective is no longer a simple mean-square reconstruction error, but rather the approximate model evidence (i.e. the marginal likelihood of model θ under the dataset $\mathcal{D}$).

Compare the loss evaluation of a classical (non-variational) autoencoder (AE):

- **Input**: data point $\mathbf{x}$
- **Encode**: $\mathbf{z} = \mathrm{NN}_\phi(\mathbf{x})$
- **Decode**: $\widehat{\mathbf{x}} = \mathrm{NN}_\theta(\mathbf{z})$
- Compute mean-square reconstruction error: $\mathrm{MSE} := \|\widehat{\mathbf{x}} - \mathbf{x}\|^2$
- **Output**: MSE

A classical AE uses a low-dimensional latent representation just like a VAE does, but the AE has no constraints on the regularity of the latent representation. Hence it is free to overfit in order to minimize the MSE on the training dataset.

8 Concluding notes

What the added complexity of a VAE buys us is that the latent representation $\mathbf{z}$ is continuous, so we can interpolate and extrapolate in the latent space. A continuous latent representation is much more useful.

The classical application are generative models. A VAE decoder can be used as a standalone generative model to produce new data statistically similar to the training inputs. This has applications in computer-generated visual art.

Exploring the latent space visually may allow a researcher to discover what the automatically extracted features represent. For datasets where the relevant features are not clear to a human *a priori*, this may lead to interesting insights, or may at least serve as a catalyst for thinking that may produce such insights.

A particularly interesting application of VAEs is in computational science, where they hold promise for acceleration of physical simulations. Because the latent representation is continuous, it should be possible to use it as a reduced order model for a dynamic simulation based on a partial differential equation model, by training another neural network to act as a time-evolution operator on the VAE-coded latent space. In a continuous coded representation, time evolution over a short timestep Δt should correspond to a short move in the latent space, which should be learnable.

Several groups at the ECCOMAS 2022 conference indeed reported successfully using autoencoders (often based on a ResNet architecture) for model order reduction. However, at this writing, this particular variant of the idea remains to be tested.

Finally, summary of some important points:

- For the low-dimensional latent space where the code points $\mathbf{z}$ live, we choose an arbitrary prior distribution $p_\theta(\mathbf{z})$, with no parameters. This is a soft constraint, specifying how we expect the codes to look like, before seeing any data.
- The distribution family used for the inference model (i.e. approximate posterior) $q_\phi(\mathbf{z}|\mathbf{x})$ constrains how the VAE can update its belief of the codes, conditioned on the data $\mathbf{x}$.
  - The result of the encoding a data point $\mathbf{x}$ is a distribution of plausible code points that could correspond to that data point.
  - The ELBO optimization will actually push the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ toward the prior $p_\theta(\mathbf{z})$.
- The distribution family used for the observation model $p_\theta(\mathbf{x}|\mathbf{z})$ reflects the distribution we expect the data to follow. Given a code point $\mathbf{z}$, it constrains how the corresponding decoded data can look like.
  - True to the stochastic nature of the VAE, when given a code point $\mathbf{z}$, the decoder (observation model) in fact yields a distribution of plausible data points that could have generated that code point. It is customary to return the mean of this distribution as the decoded $\mathbf{x}$, but one could also draw a sample, or return the distribution itself.
- The mapping between the data and code spaces *is not explicitly specified*.
  - This mapping — in the form of parameters for pre-chosen families of probability distributions $p_\theta(\mathbf{x}|\mathbf{z})$ and $q_\phi(\mathbf{z}|\mathbf{x})$ — is discovered by the neural networks $\mathrm{NN}_\theta(\mathbf{z})$ and $\mathrm{NN}_\phi(\mathbf{x})$, by jointly training them against the dataset $\mathcal{D}$ via the ELBO loss. This neural network optimization, together with the low-dimensional bottleneck, causes the latent space — which we forced in itself to have a simple structure — to become a representation for features discovered from the data. The "magic" is in the neural networks.

9 References

Diederik P Kingma, Max Welling. 2013 (revised 2022). Auto-Encoding Variational Bayes. https://arxiv.org/abs/1312.6114

Diederik P Kingma, Max Welling. 2019. An Introduction to Variational Autoencoders. https://arxiv.org/abs/1906.02691

Gabriel Loaiza-Ganem, John P. Cunningham. 2019. The continuous Bernoulli: fixing a pervasive error in variational autoencoders. https://arxiv.org/abs/1907.06845

Shuyu Lin, Stephen Roberts, Niki Trigoni, Ronald Clark. 2019. Balancing Reconstruction Quality and Regularisation in ELBO for VAEs. https://arxiv.org/abs/1909.03765

10 Online resources for further reading

Clicky clicky. Currently a link dump without much regard to formatting; may organize later.

Also the cited papers are highly recommended, for providing important ideas in a clearly written form that is easy to follow.

VAE-related
https://www.deeplearningbook.org/contents/generative_models.html (the Deep Learning book by Goodfellow et al. (2016) includes a section on VAEs)
https://danijar.com/building-variational-auto-encoders-in-tensorflow/
https://www.cs.toronto.edu/~frossard/post/vgg16/ (excellent network structure diagram)
https://wizardforcel.gitbooks.io/tensorflow-examples-aymericdamien/content/3.10_variational_autoencoder.html
https://linux-blog.anracom.com/2022/10/06/variational-autoencoder-with-tensorflow-2-
https://linux-blog.anracom.com/2022/10/22/variational-autoencoder-with-tensorflow-2-
https://linux-blog.anracom.com/2022/11/07/variational-autoencoder-with-tensorflow-2-
https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919
https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd
https://towardsdatascience.com/bayesian-inference-problem-mcmc-and-variational-infere
https://learnopencv.com/autoencoder-in-tensorflow-2-beginners-guide/
https://learnopencv.com/variational-autoencoder-in-tensorflow/
https://www.inference.vc/choice-of-recognition-models-in-vaes-a-regularisation-view/
https://www.inference.vc/variational-inference-using-implicit-models/
https://www.inference.vc/how-to-train-your-generative-models-why-generative-adversar

Basic concepts
https://www.baeldung.com/cs/k-fold-cross-validation
https://machinelearningmastery.com/difference-test-validation-datasets/
https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-fr
https://www.quora.com/What-is-the-difference-between-skip-peephole-and-residual-conn
https://andrewcharlesjones.github.io/journal/convergence.html (convergence in probability vs. almost sure conv.)
https://github.com/y0ast/VAE-TensorFlow/issues/3 (why Bernoulli distribution in VAE decoder)
https://mbernste.github.io/posts/elbo/

Backpropagation (i.e. backward-mode automatic differentiation for neural networks)
https://cs231n.github.io/optimization-1/#gd
https://cs231n.github.io/neural-networks-case-study/#grad
https://dustinstansbury.github.io/theclevermachine/derivation-backpropagation
https://karpathy.medium.com/yes-you-should-understand-backprop-e2f06eab496b

Various AI/NN tutorials
https://dennybritz.com/posts/wildml/implementing-a-neural-network-from-scratch/
https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-4/
https://mkffl.github.io/2019/07/08/minimalist-RNN.html
https://colah.github.io/posts/2015-08-Understanding-LSTMs/
https://kvfrans.com/variational-autoencoders-explained/

https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798
https://spinningup.openai.com/en/latest/user/introduction.html (reinforcement learning tutorial)
https://neptune.ai/blog/reinforcement-learning-agents-training-debug (how to debug RL)

Series on high-resolution image generation from examples
https://blog.otoro.net/2016/03/25/generating-abstract-patterns-with-tensorflow/ (CPPN)
https://blog.otoro.net/2016/04/01/generating-large-images-from-latent-vectors/ (VAE + GAN)
https://blog.otoro.net/2016/06/02/generating-large-images-from-latent-vectors-part-tw
(condition to diversify)

Wikipedia
https://en.wikipedia.org/wiki/Training%2C_validation%2C_and_test_data_sets
https://en.wikipedia.org/wiki/Logistic_distribution
https://en.wikipedia.org/wiki/Logit
https://en.wikipedia.org/wiki/Bernoulli_distribution
https://en.wikipedia.org/wiki/Continuous_Bernoulli_distribution (new, 2019!)
https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding
https://en.wikipedia.org/wiki/Residual_neural_network (skip-connections)
https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence
https://en.wikipedia.org/wiki/Compositional_pattern-producing_network (how to teach AI abstract visual art)
https://en.wikipedia.org/wiki/Neuroevolution

Web demos
https://laion.ai/blog/laion-5b/
https://stablediffusionweb.com/
https://prostheticknowledge.tumblr.com/post/136696656421/dcgan-face-generator-online-
https://qiita.com/mattya/items/e5bfe5e04b9d2f0bbd47 (Chainer - generate anime portraits via GAN)

Code examples
https://blog.fastforwardlabs.com/2016/02/24/hello-world-in-keras-or-scikit-learn-vers
html
https://keras.io/examples/generative/vae/
https://www.tensorflow.org/tutorials/generative/autoencoder
https://www.tensorflow.org/tutorials/generative/cvae
https://github.com/ChengBinJin/VAE-Tensorflow/tree/master/src
https://github.com/cunningham-lab/cb_and_cc/blob/master/cb/norm_vae_mnist.ipynb
https://github.com/cunningham-lab/cb_and_cc/blob/master/cb/utils.py (the original continuous Bernoulli implementation)
https://github.com/farrell236/ResNetAE/blob/master/ResNetAE.py
https://github.com/openai/improved-gan
https://github.com/wxs/keras-mnist-tutorial/blob/master/MNIST%20in%20Keras.ipynb
https://www.tensorflow.org/guide/keras/custom_layers_and_models#putting_it_all_

together_an_end-to-end_example
https://www.nltk.org/
https://github.com/pietrobarbiero/pytorch_explain
https://github.com/kvfrans/variational-autoencoder
https://jmetzen.github.io/2015-11-27/vae.html

API docs
https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50
https://simplegan.readthedocs.io/en/latest/modules/autoencoder.html
https://keras.io/api/layers/convolution_layers/convolution2d/
https://keras.io/api/layers/convolution_layers/convolution2d_transpose/
https://developer.nvidia.com/blog/optimizing-gpu-performance-tensor-cores/
https://docs.nvidia.com/deeplearning/frameworks/tensorflow-user-guide/index.html

Various blog posts
https://karpathy.github.io/2015/05/21/rnn-effectiveness/
https://gist.github.com/karpathy/d4dee566867f8291f086 (code for above; minimal character-level RNN in NumPy)
https://towardsdatascience.com/classification-model-for-source-code-programming-langu

Various AI-related code snippets
https://github.com/stratospark/keras-multiprocess-image-data-generator
https://github.com/bckenstler/CLR (cyclical learning rate plugin for Keras)