Auto-encoders: reconstruction versus compression

Yann Ollivier

Abstract

We discuss the similarities and differences between training an auto-encoder to minimize the reconstruction error, and training the same auto-encoder to compress the data via a generative model. Minimizing a codelength for the data using an auto-encoder is equivalent to minimizing the reconstruction error plus some correcting terms which have an interpretation as either a *denoising* or *contractive* property of the decoding function. These terms are related but not identical to those used in denoising or contractive auto-encoders [VLL+10, RVM+11]. In particular, the codelength viewpoint fully determines an optimal noise level for the denoising criterion.

Given a dataset, auto-encoders (for instance, [PH87, Section 8.1] or [HS06]) aim at building a hopefully simpler representation of the data via a hidden, usually lower-dimensional *feature space*. This is done by looking for a pair of maps $X \xrightarrow{f} Y \xrightarrow{g} X$ from data space X to feature space Y and back, such that the reconstruction error between x and $g(f(x))$ is small. Identifying relevant features hopefully makes the data more understandable, more compact, or simpler to describe.

Here we take this interpretation literally, by considering auto-encoders in the framework of minimum description length (MDL), i.e., data compression via a probabilistic generative model, using the general correspondence between compression and "simple" probability distributions on the data [Grü07]. The objective is then to minimize the codelength (log-likelihood) of the data using the features found by the auto-encoder[1].

We use the "variational" approach to answer the following question: Do auto-encoders trained to minimize reconstruction error actually minimize the length of a compressed encoding of the data, at least approximately?

We will see that by adding an information-theoretic term to the reconstruction error, auto-encoders can be trained to minimize a tight upper bound on the codelength (compressed size) of the data.

In Section 3 we introduce a first, simple bound on codelength based on reconstruction error: a dataset $\mathcal{D} \subset X$ can be encoded by encoding a

[1]The goal here is not to build an actual compressed code of the data, but to find a good pair of feature and generative functions that *would* yield a short codelength [Grü07]. If the codelength is known as a function of the parameters of the auto-encoder, it can be used as the training criterion.

(hopefully simpler) feature value $f(x)$ for each $x \in \mathcal{D}$, and applying the decoding function g. However, this result only applies to discrete features, and the resulting bound is far from tight. Still, this already illustrates how minimizing codelength favors using fewer features.

In Section 4 we refine the bound from Section 3 and make it valid for general feature spaces (Proposition 2). This bound is tight in the sense that it gets arbitrarily close to the actual codelength when the feature and generative functions are inverse to each other in a probabilistic sense. This is an instance of the *variational bound* [Bis06, Chapter 10]. Related results appear in [HOT06] and in [KW13, Section 2.2].

The result in Section 4 also illustrates how, to optimize codelength, an auto-encoder approach helps compared to directly looking for a generative model. Trying to optimize the codelength directly is often difficult (Section 2). So even though the codelength L_{gen} depends only on the generative function g and not on a feature function, we build an upper bound on L_{gen} depending on both; optimizing over g aims at lowering L_{gen} by lowering this upper bound, while optimizing over f aims at making the upper bound more precise.

In Sections 5 and 6 we provide a connection with *denoising auto-encoders* [VLL+10]. When the feature space is continuous, it is impossible to encode a feature value $f(x)$ exactly for each x in the dataset as this yields an infinite codelength. Thus, it is necessary to encode features with finite precision and to use a decoding function that is not too sensitive to approximate features. Quantifying this effect leads to an explicit upper bound on codelength (Corollary 3). The denoising criterion is from features to output, rather than from input to features as in [VLL+10].

Moreover the MDL approach allows us to find the optimal noise level for the denoising criterion, i.e., the one which yields the best codelength (Theorem 5). In particular, the noise level should be set differently for each data sample.

In Section 7 we establish a connection with *contractive auto-encoders* [RVM+11]: under various approximations, minimizing codelength penalizes large derivatives of the output (Proposition 6). The penalty takes a form somewhat different from [RVM+11], though: contractivity occurs from features to output rather than from input to features, and the penalty term is not the Frobenius norm of the Jacobian matrix but the sum of the logs of the norms of its rows. An advantage of the MDL approach is that the penalty constant is determined from theory.

In Section 8 we show that optimal compression requires including the variance of each data component as additional parameters, especially when various data components have different variances or noise levels. Compression focuses on relative rather than absolute error, minimizing the *logarithms* of the errors.

The variational bound has already been applied to neural networks in non-auto-encoding situations, to evaluate the cost of encoding the network

parameters [Gra11, HvC93]. In that situation, one tries to find a map $Y \stackrel{g}{\to} X$ that minimizes the codelength of the output data x *if the features* y *are given*; this decomposes as the output error plus a term describing the cost of encoding the parameters of g. In an auto-encoding setting $X \stackrel{f}{\to} Y \stackrel{g}{\to} X$, it is meaningless to encode the dataset given the very same inputs: so the dataset is encoded by encoding the features y together with g. In this text we focus on the cost of encoding y, and the consequences of minimizing the resulting codelength. Encoding of the parameters of g can be done following [Gra11] and we do not reproduce it here. Still, the cost of g must be included for actual data compression, and also especially when comparing generative models with different dimensions.

1 Notation: Auto-encoders, reconstruction error. Let X be an input space and Y be a feature space, usually of smaller dimension. Y may be discrete, such as $Y = \{0,1\}^d$ (each feature present/absent) or $Y = \{1, \dots, d\}$ (classification), or continuous.

An auto-encoder can be seen as a pair of functions f and g, the *feature* function and the *generative* function. The feature function goes from X to Y (deterministic features) or to $\mathrm{Prob}(Y)$ (probability distribution on features), while the generative function goes from Y to X or $\mathrm{Prob}(X)$.

The functions f and g depend on parameters θ_f and θ_g respectively. For instance, f and g may each represent a multilayer neural network or any other model. Training the parameters via the reconstruction error criterion focuses on having $g(f(x))$ close to x, as follows.

Given a feature function $f\colon X \to Y$ and a generative function $g\colon Y \to \mathrm{Prob}(X)$, define the *reconstruction error* for a dataset $\mathcal{D} \subset X$ as

$$L_{\mathrm{rec}}(x) := -\log g_{f(x)}(x), \qquad L_{\mathrm{rec}}(\mathcal{D}) := \sum_{x \in \mathcal{D}} L_{\mathrm{rec}}(x) \tag{1}$$

where g_y is the probability distribution on X associated with feature y.

The case of a deterministic $g\colon Y \to X$ with square error $\|g(f(x)) - x\|^2$ is recovered by interpreting g as a Gaussian distribution[2] centered at $g(f(x))$. So we will always consider that g is a probability distribution on X.

Discrete-valued features can be difficult to train using gradient-based methods. For this reason, with discrete features it is more natural to define $f(x)$ as a distribution over the feature space Y describing the law of inferred features for x. Thus $f(x)$ will have continuous parameters. If $f\colon X \to \mathrm{Prob}(Y)$ describes a probability distribution on features for each x, we define the expected reconstruction error as the expectation of the above:

$$\mathbb{E}L_{\mathrm{rec}}(x) := -\mathbb{E}_{y \sim f(x)} \log g_y(x), \qquad \mathbb{E}L_{\mathrm{rec}}(\mathcal{D}) := \sum_{x \in \mathcal{D}} \mathbb{E}L_{\mathrm{rec}}(x) \tag{2}$$

[2]While the choice of variance does not influence minimization of the reconstruction error, when working with codelengths it will change the scaling of the various terms in Propositions 1–6. See Section 8 for the optimal variance

This covers the previous case when $f(x)$ is a Dirac mass at a single value y.

In Sections 2–4 the logarithms may be in any base; in Sections 5–8 the logarithms are in base e.

2 Auto-encoders as generative models. Alternatively, auto-encoders can be viewed as generative models for the data. For this we assume that we are given (or learn) an elementary model ρ on feature space, such as a Gaussian or Bernoulli model, or even a uniform model in which each feature is present or absent with probability 1/2. Then, to generate the data, we draw features at random according to ρ and apply the generative function g. The goal is to maximize the probability to generate the actual data. In this viewpoint the feature function f is used only as a prop to learn a good feature space and a good generative function g.

Given a probability distribution p on a set X, a dataset $(x_1, \dots, x_n)$ of points on X can be encoded in $-\sum_i \log_2 p(x_i)$ bits[3]. Let $\rho \in \mathrm{Prob}(Y)$ be the elementary model on feature space and let $g\colon Y \to \mathrm{Prob}(X)$ be the generative function. The probability to obtain $x \in X$ by drawing $y \sim \rho$ and applying g is

$$p_g(x) := \int_y \rho(y) g_y(x) \tag{3}$$

(where the integral is a sum if the feature space Y is discrete). Thus minimizing the codelength of the dataset $\mathcal{D}$ amounts to minimizing

$$L_{\mathrm{gen}}(\mathcal{D}) := \sum_{x \in \mathcal{D}} L_{\mathrm{gen}}(x), \tag{4}$$

$$L_{\mathrm{gen}}(x) := -\log p_g(x) = -\log \int_y \rho(y) g_y(x) \tag{5}$$

over g.

This is the codelength of the data knowing the distribution ρ and the function g. We do not consider here the problem of encoding the parameters of ρ and g; this can be done following [Gra11], for instance.

The codelength L_{gen} does not depend on any feature function f. However, it is difficult to optimize L_{gen} via a direct approach: this leads to working with all possible values of y for every sample x, as $L_{\mathrm{gen}}(x)$ is an integral over y. Presumably, for each given x only a few feature values contribute significantly to $L_{\mathrm{gen}}(x)$. Using a feature function is a way to explore fewer possible values of y for a given x, hopefully those that contribute most to $L_{\mathrm{gen}}(x)$.

[3]Technically, for continuous-valued data x, the actual compressed length is rather $-\log_2 p(x) - \log_2 \varepsilon$ where ε is the quantization threshold of the data and p is the probability density for x. For the purpose of comparing two different probabilistic models p on the same data with the same ε, the term $-\log_2 \varepsilon$ can be dropped

For instance, consider the gradient of $L_{\text{gen}}(x)$ with respect to a parameter θ:

$$\frac{\partial L_{\text{gen}}(x)}{\partial \theta} = -\frac{\int_y \rho(y)\partial g_y(x)/\partial\theta}{\int_y \rho(y) g_y(x)} = -\frac{\int_y \rho(y) g_y(x)\partial \ln g_y(x)/\partial\theta}{\int_y \rho(y) g_y(x)} \tag{6}$$

$$= -\mathbb{E}_{y\sim p_g(y|x)} \frac{\partial \ln g_y(x)}{\partial\theta} \tag{7}$$

where $p_g(y|x) = \rho(y)g_y(x) \,/\, \int_{y'} \rho(y')g_{y'}(x)$ is the conditional probability of y knowing x, in the generative model given by ρ and g. In general we have no easy access to this distribution.

Using a (probabilistic) feature function f and minimizing the reconstruction error $\mathbb{E}L_{\text{rec}}(x)$ amounts to replacing the expectation under $y \sim p_g(y|x)$ with an expectation under $f(x)$ in the above, presumably easier to handle. However this gives no guarantees about minimizing L_{gen} unless we know that the feature function f is close to the inverse of the generative function g, in the sense that $f(x)(y)$ is close to the conditional distribution $p_g(y|x)$ of y knowing x. It would be nice to obtain a guarantee on the codelength based on the reconstruction error of a feature function f and generative function g.

The variational bound in Proposition 2 below shows that, given a feature function f and a generative function g, the quantity $L_{\text{rec}}(x) + \mathrm{KL}(f(x) \,\|\, \rho)$ is an upper bound on the codelength $L_{\text{gen}}(x)$. Training an autoencoder to minimize this criterion will thus minimize an upper bound on L_{gen}.

Moreover, Proposition 2 shows that the bound is tight when $f(x)$ is close to $p_g(y|x)$, and that minimizing this bound will indeed bring $f(x)$ closer to $p_g(y|x)$. On the other hand, just minimizing the reconstruction error does not, a priori, guarantee any of this.

3 Two-part codes: explicitly encoding feature values. We first discuss a simple, less efficient "two-part" [Grü07] coding method. It always yields a codelength larger than L_{gen} but is more obviously related to the auto-encoder reconstruction error.

Given a generative model $g\colon Y \to \mathrm{Prob}(X)$ and a prior[4] distribution ρ on Y, one way to encode a data sample $x \in X$ is to explicitly encode a well-chosen feature value $y \in Y$ using the prior distribution ρ on features, then encode x using the probability distribution $g_y(x)$ on x defined by y. The codelength resulting from this choice of y is thus

$$L_{\text{two-part}}(x) := -\log \rho(y) - \log g_y(x) \tag{8}$$

In this section we assume that Y is a discrete set. Indeed for continuous features, the above does not make sense as encoding a precise value for y

[4]We use the term "prior" in a loose way: we just encode features y with a code of length $-\log \rho(y)$, without implying any a priori belief. Thus ρ is just a simple model used on feature space.

would require an infinite codelength. Continuous features are dealt with in Sections 4 and 5.

We always have

$$L_{\text{two-part}}(x) \geqslant L_{\text{gen}}(x) \tag{9}$$

for discrete features, as $L_{\text{two-part}}$ uses a single value of y while L_{gen} uses a sum over y. The difference can be substantial if, for instance, not all feature components are relevant for all x: using the two-part code, it is always necessary to fully encode the feature values y.

From an auto-encoder perspective, the feature function f is used to choose the feature value y used to encode x. So if the feature function is deterministic, $f\colon X \to Y$, and if we set $y = f(x)$ in the above, the cost of encoding the dataset is

$$\begin{aligned} L_{\text{two-part}}(\mathcal{D}) &= -\sum_{x\in\mathcal{D}} \Big(\log \rho(f(x)) + \log g_{f(x)}(x)\Big) \\ &= L_{\text{rec}}(\mathcal{D}) - \sum_{x\in\mathcal{D}} \log \rho(f(x)) \end{aligned}$$

involving the reconstruction error and a cross-entropy term between the empirical distribution of features $f(x)$ and the prior ρ on feature space. We can further decompose

$$-\frac{1}{\#\mathcal{D}} \sum_{x\in\mathcal{D}} \log \rho(f(x)) = \mathrm{KL}(q_f \,\|\, \rho) + \mathrm{Ent}\, q_f \tag{10}$$

where q_f is the empirical distribution of the feature $f(x)$ when x runs over the dataset,

$$q_f(y) := \frac{1}{\#\mathcal{D}} \sum_{x\in\mathcal{D}} \mathbb{1}_{f(x)=y} \tag{11}$$

and $\mathrm{KL}(q_f \,\|\, \rho) = \mathbb{E}_{y\sim q_f} \log(q_f(y)/\rho(y))$ is the Kullback–Leibler divergence between q_f and ρ.

If the feature function f is probabilistic, $f\colon X \to \mathrm{Prob}(Y)$, the analysis is identical, with the expected two-part codelength of x being

$$\mathbb{E}L_{\text{two-part}}(x) = \mathbb{E}_{y\sim f(x)}(-\log \rho(y) - \log g_y(x)) \tag{12}$$

$$= \mathbb{E}L_{\text{rec}}(x) - \mathbb{E}_{y\sim f(x)} \log \rho(y) \tag{13}$$

Thus we have proved the following, which covers both the case of probabilistic f and of deterministic f (by specializing f to a Dirac mass) on a discrete feature space.

Proposition 1 (Two-part codelength and reconstruction error for discrete features). *The expected two-part codelength*

of $x \in \mathcal{D}$ and the reconstruction error are related by

$$\mathbb{E}L_{\text{two-part}}(\mathcal{D}) = \mathbb{E}L_{\text{rec}}(\mathcal{D}) - \sum_{x\in\mathcal{D}} \mathbb{E}_{y\sim f(x)} \log \rho(y) \tag{14}$$

$$= \mathbb{E}L_{\text{rec}}(\mathcal{D}) + (\#\mathcal{D})(\mathrm{KL}(q_f \,\|\, \rho) + \operatorname{Ent} q_f) \tag{15}$$

where

$$q_f(y) := \frac{1}{\#\mathcal{D}} \sum_{x\in\mathcal{D}} \Pr(f(x) = y) \tag{16}$$

is the empirical distribution of features.

Here are a few comments on this relation. These comments also apply to the codelength discussed in Section 4.

- The reconstruction error in (14) is the *average* reconstruction error for features y sampled from $f(x)$, in case $f(x)$ is probabilistic. For instance, applying Proposition 1 to neural networks requires interpreting the activities of the Y layer as probabilities to sample 0/1-valued features on the Y layer. (This is not necessary for the results of Sections 4–7, which hold for continuous features.)

- The cross-entropy term $-\mathbb{E}_{y\sim q_f} \log \rho(y) = \mathrm{KL}(q_f \,\|\, \rho) + \operatorname{Ent} q_f$ is an added term to the optimisation problem. The Kullback–Leibler divergence favors feature functions that do actually match an elementary model on Y, e.g., feature distributions that are "as Bernoulli-like" as possible. The entropy term $\operatorname{Ent} q_f$ favors parsimonious feature functions that use fewer feature components if possible, arguably introducing some regularization or sparsity. (Note the absence of any arbitrary parameter in front of this regularization term: its value is fixed by the MDL interpretation.)

- If the elementary model ρ has tunable parameters (e.g., a Bernoulli parameter for each feature), these come into the optimization problem as well. If ρ is elementary it will be fairly easy to tune the parameters to find the elementary model $\rho^*(f)$ minimizing the Kullback–Leibler divergence to q_f. Thus in this case the optimization problem over f and g involves a term $\mathrm{KL}(q_f \,\|\, \rho^*(f))$ between the empirical distribution of features and the closest elementary model.

This two-part code is somewhat naive in case not all feature components are relevant for all samples x: indeed for every x, a value of y has to be fully encoded. For instance, with feature space $Y = \{0,1\}^d$, if two values of y differ in one place and contribute equally to generating some sample x, one could expect to save one bit on the codelength, by leaving a blank in the encoding where the two values of y differ. In general, one could expect to save $\operatorname{Ent} f(x)$ bits on the encoding of y if several $y \sim f(x)$ have a high probability to generate x. We now show that indeed $\mathbb{E}L_{\text{two-part}}(x) - \operatorname{Ent} f(x)$ is still an upper bound on $L_{\text{gen}}(x)$.

4 Comparing L_{gen} and L_{rec}. We now turn to the actual codelength $L_{\text{gen}}(x) = -\log p_g(x)$ associated with the probabilistic model $p_g(x)$ defined by the generative function g and the prior ρ on feature space. As mentioned above, it is always smaller that the two-part codelength.

Recall that this model first picks a feature value y at random according to the distribution ρ and then generates an object x according to the distribution $g_y(x)$, so that the associated codelength is $-\log p_g(x) = -\log \int_y \rho(y) g_y(x)$.

So far this does not depend on the feature function so it is not clear how f can help in optimizing this codelength. Actually each choice of f leads to upper bounds on L_{gen}: the two-part codelength $L_{\text{two-part}}$ above is one such bound in the discrete case, and we now introduce a more precise and more general one, $L_{f\text{-gen}}$.

We have argued above (Section 2) that for gradient-based training it would be helpful to be able to sample features from the distribution $p_g(y|x)$, and it is natural to expect the feature function $f(x)$ to approximate $p_g(y|x)$, so that f and g are inverse to each other in a probabilistic sense. The tightness of the bound $L_{f\text{-gen}}$ is related to the quality of this approximation. Moreover, while auto-encoder training based on the reconstruction error provides no guarantee that f will get closer to $p_g(y|x)$, minimizing $L_{f\text{-gen}}$ does.

Proposition 2 (Codelength and reconstruction error for probabilistic features). *The codelength L_{gen} and reconstruction error L_{rec} for an auto-encoder with feature function $f\colon X \to \text{Prob}(Y)$ and generative function $g\colon Y \to \text{Prob}(X)$ satisfy*

$$L_{\text{gen}}(x) = \mathbb{E}L_{\text{rec}}(x) + \text{KL}(f(x) \,\|\, \rho) - \text{KL}(f(x) \,\|\, p_g(y|x))\,, \tag{17}$$

$$L_{\text{gen}}(\mathcal{D}) = \mathbb{E}L_{\text{rec}}(\mathcal{D}) + \sum_{x\in\mathcal{D}} \text{KL}(f(x) \,\|\, \rho) - \sum_{x\in\mathcal{D}} \text{KL}(f(x) \,\|\, p_g(y|x)) \tag{18}$$

where ρ is the elementary model on features, and $p_g(y|x) = \frac{\rho(y) g_y(x)}{\int_{y'} \rho(y') g_{y'}(x)}$.

In particular, for any feature function f, the quantity

$$L_{f\text{-gen}}(\mathcal{D}) := \sum_{x\in\mathcal{D}} L_{f\text{-gen}}(x) \tag{19}$$

where

$$L_{f\text{-gen}}(x) := \mathbb{E}L_{\text{rec}}(x) + \text{KL}(f(x) \,\|\, \rho) \tag{20}$$

is an upper bound on the codelength $L_{\text{gen}}(\mathcal{D})$ of the generative function g.

The result holds whether Y is discrete or continuous.

The proof is by substitution in the right-hand-side of (17); actually this is an instance of the *variational bound* [Bis06, Chapter 10]. Related results appear in [HOT06] and [KW13, Section 2.2].

On a discrete feature space, $L_{f\text{-gen}}$ is always smaller than the codelength $L_{\text{two-part}}$ above; indeed

$$L_{f\text{-gen}}(x) = \mathbb{E}L_{\text{two-part}}(x) - \operatorname{Ent} f(x) \tag{21}$$

as can be checked directly.

The term $\mathrm{KL}(f(x) \,\|\, \rho)$ represents the cost of encoding a feature value y drawn from $f(x)$ for each x (encoded using the distribution ρ). The last, negative term in (17)–(18) represents how pessimistic the reconstruction error is w.r.t. the true codelength when $f(x)$ is far from the feature values that contribute most to $L_{\text{gen}}(x)$.

The codelength L_{gen} depends only on g and not on the feature function f, so that the right-hand-side in (17)–(18) is the same for all f despite appearances. Ideally, this relation could be used to evaluate $L_{\text{gen}}(\mathcal{D})$ for a given generative function g, and then to minimize this quantity over g. However, as explained above, the conditional probabilities $p_g(y|x)$ are not easy to work with, hence the introduction of the upper bound $L_{f\text{-gen}}$, which does depend on the feature function f.

Minimizing $L_{f\text{-gen}}$ over f will bring $L_{f\text{-gen}}$ closer to L_{gen}. Since $L_{\text{gen}}(\mathcal{D}) = L_{f\text{-gen}}(\mathcal{D}) - \sum_{x\in\mathcal{D}} \mathrm{KL}(f(x) \,\|\, p_g(y|x))$, and since L_{gen} does not depend on f, minimizing $L_{f\text{-gen}}$ is the same as bringing $f(x)$ closer to $p_g(y|x)$ on average. Thus, in the end, an auto-encoder trained by minimizing $L_{f\text{-gen}}$ as a function of f and g will both minimize an upper bound on the codelength L_{gen} and bring $f(x)$ close to the "inverse" of g.

This also clarifies the role of the auto-encoder structure in minimizing the codelength, which does not depend on a feature function: Optimizing over g aims at actually reducing the codelength by decreasing an upper bound on it, while optimizing over f will make this upper bound more precise.

One can apply to $L_{f\text{-gen}}$ the same decomposition as for the two-part codelength, and write

$$L_{f\text{-gen}}(\mathcal{D}) = \mathbb{E}L_{\text{rec}}(\mathcal{D}) + (\#\mathcal{D})(\mathrm{KL}(q_f \,\|\, \rho) + \operatorname{Ent} q_f) - \sum_{x\in\mathcal{D}} \operatorname{Ent} f(x) \tag{22}$$

where as above $q_f = \frac{1}{\#\mathcal{D}} \sum_{x\in\mathcal{D}} f(x)$ is the empirical feature distribution. As above, the term $\mathrm{KL}(q_f \,\|\, \rho)$ favors feature distributions that match a simple model. The terms $\operatorname{Ent} q_f$ and $\sum_{x\in\mathcal{D}} \operatorname{Ent} f(x)$ pull in different directions. Minimizing $\operatorname{Ent} q_f$ favors using fewer features overall (more compact representation). Increasing the entropy of $f(x)$ for a given x, if it can be done without impacting the reconstruction error, means that more features are "indifferent" for reconstructing x and do not have to be encoded, as discussed at the end of Section 3.

The "auto-encoder approximation" $x' = x$ from [AO12, Section 2.4] can be used to define another bound on L_{gen}, but is not tight when $f(x) \approx p_g(y|x)$.

5 Continuous-valued features and denoising. Proposition 2 cannot be directly applied to a deterministic feature function $f\colon X \to Y$ with values in a continuous space Y. In the continous case, the reconstruction error based on a single value $y \in Y$ cannot control the codelength $L_{\text{gen}}(x)$, which involves an integral over y. In the setting of Proposition 2, a deterministic f seen as a probability distribution is a Dirac mass at a single value, so that the term $\mathrm{KL}(f(x) \,\|\, \rho)$ is infinite: it is infinitely costly to encode the feature value $f(x)$ exactly.

This can be overcome by considering the feature values y as probability distributions over an underlying space Z, namely, $Y \subset \mathrm{Prob}(Z)$. Then Proposition 2 can be applied to $f(x)$ seen as a probability distribution over the feature space Z.

For instance, one possibility for neural networks with logistic activation function is to see the activities $y \in [0;1]$ of the feature layer as Bernoulli probabilities over discrete-valued binary features, $Z = \{0,1\}$.

One may also use Gaussian distributions over $Z = Y$ and apply Proposition 2 to a normal distribution $\mathcal{N}(f(x), \Sigma)$ centered at $f(x)$ with small covariance matrix Σ. Intuitively we overcome the problem of infinite codelength for $f(x)$ by encoding $f(x)$ with finite accuracy given by Σ.

The reconstruction error L_{rec} from Proposition 2 then becomes an expectation over features sampled around $f(x)$: this is similar to *denoising* auto-encoders [VLL+10], except that here the noise is added to the features rather than the inputs. This relationship is not specific to a particular choice of feature noise (Bernoulli, Gaussian...) but leads to interesting developments in the Gaussian case, as follows.

Corollary 3 (Codelength and denoising the features). *Let $f\colon X \to Y$ be a deterministic feature function with values in $Y = \mathbb{R}^d$. Let Σ be any positive definite matrix. Then*

$$L_{\text{gen}}(x) \leqslant \mathbb{E}L_{\text{rec}}(x) - \mathbb{E}_{y\sim\mathcal{N}(f(x),\Sigma)} \log \rho(y) - \frac{1}{2}\log\det\Sigma - \frac{d}{2}(1+\log 2\pi) \quad (23)$$

where $\mathbb{E}L_{\text{rec}}(x)$ is the expected reconstruction error obtained from a feature $y \sim \mathcal{N}(f(x), \Sigma)$.

If the elementary model ρ on feature space is $\mathcal{N}(0, \lambda\,\mathrm{Id})$ this reads

$$L_{\text{gen}}(x) \leqslant \mathbb{E}L_{\text{rec}}(x) + \frac{\|f(x)\|^2}{2\lambda} + \frac{1}{2\lambda}\,\mathrm{Tr}(\Sigma) - \frac{1}{2}\log\det\Sigma + \frac{d}{2}\log\lambda - \frac{d}{2} \quad (24)$$

Thus the codelength bound decomposes as the sum of the average (noisy) reconstruction error, constant terms, and a term that penalizes improbable feature values under the elementary model.

Proof.
Apply Proposition 2 with a normal distribution $\mathcal{N}(f(x), \Sigma)$ as the feature distribution. □

We refer to [VLL+10, Section 4.2] for a discussion and further references on training with noise in an auto-encoder setting.

In practice, the bound (23) can be optimized over f and g via Monte Carlo sampling over $y \sim \mathcal{N}(f(x), \Sigma)$. For the case of neural networks, this can be done via ordinary backpropagation if one considers that the activation function of the layer representing Y is $y = f(x) + \mathcal{N}(0, \Sigma)$: one can then run several independent samples y_i, backpropagate the loss obtained with each y_i, and average over i. The backpropagation from y to the input layer can even be factorized over the samples, thanks to linearity of backpropagation, namely: generate samples $y_i \sim \mathcal{N}(f(x), \Sigma)$, backpropagate the error obtained with y_i from the output to the layer representing Y, average the obtained backpropagated values over i, and backpropagate from the Y layer to the input layer using f. For any explicit choice of ρ, the contribution of the gradient of the $\log \rho(y)$ term can easily be incorporated into this scheme.

6 Optimal noise level. A good choice of noise level Σ leads to tighter bounds on L_{gen}: a small Σ results in a high cost of encoding the features up to Σ ($\log\det\Sigma$ term), while a large Σ will result in more noise on features and a worse reconstruction error. An approximately optimal choice of Σ can be obtained by a Taylor expansion of the reconstruction error around $f(x)$, as follows. (A theoretical treatment of using such Taylor expansions for optimization with denoising can be found in [GCB97].)

Lemma 4 (Taylor expansion of $L_{f\text{-gen}}$ for small Σ). *Let $f\colon X \to Y$ be a deterministic feature function with values in $Y = \mathbb{R}^d$. Let $L_{f,\Sigma\text{-gen}}$ be the upper bound (23) using a normal distribution $\mathcal{N}(f(x), \Sigma)$ for features. Then for small covariance matrix Σ we have*

$$L_{f,\Sigma\text{-gen}}(x) \approx L_{\text{rec}}(x) - \log\rho(f(x)) - \frac{1}{2}\log\det\Sigma + \frac{1}{2}\operatorname{Tr}(\Sigma H) - \frac{d}{2}(1 + \log 2\pi) \tag{25}$$

where $L_{\text{rec}}(x)$ is the deterministic reconstruction error using feature $y = f(x)$, and H is the Hessian

$$H = \frac{\partial^2}{\partial y^2}(L^y_{\text{rec}}(x) - \log\rho(y)) \tag{26}$$

at $y = f(x)$, where $L^y_{\text{rec}}(x)$ is the reconstruction error using feature y. Thus this is an approximate upper bound on $L_{\text{gen}}(x)$.

Theorem 5 (Optimal choice of Σ for feature noise). *Let $f\colon X \to Y$ be a deterministic feature function with values in $Y = \mathbb{R}^d$. Let $L_{f,\Sigma\text{-gen}}$ be the upper bound (23) using a normal distribution $\mathcal{N}(f(x), \Sigma)$ for features. Let as above*

$$H(x) := \frac{\partial^2}{\partial y^2}(L^y_{\text{rec}}(x) - \log\rho(y)) \tag{27}$$

at $y = f(x)$.

Then the choice $\Sigma(x) = H(x)^{-1}$ *(provided* H *is positive) is optimal in the bound* (25) *and yields*

$$L_{f,\Sigma\text{-gen}}(x) \approx L_{\text{rec}}(x) - \log \rho(f(x)) + \frac{1}{2} \log \det H(x) - \frac{d}{2} \log 2\pi \tag{28}$$

as an approximate upper bound on $L_{\text{gen}}(x)$.

Among diagonal matrices Σ*, the optimal choice is* $\Sigma(x) = (\operatorname{diag} H(x))^{-1}$ *and produces a corresponding term* $\frac{1}{2} \log \det \operatorname{diag} H(x)$ *instead of* $\frac{1}{2} \log \det H(x)$.

In addition to the reconstruction error $L_{\text{rec}}(x)$ at $f(x)$ and to the encoding cost $-\log \rho(f(x))$ under the elementary model, this codelength bound involves the reconstruction error around $f(x)$ through the Hessian. Minimizing this bound will favor points where the error is small in the widest possible feature region around $f(x)$. This presumably leads to more robust reconstruction.

Several remarks can be made on this result. First, the optimal choice of noise Σ depends on the data sample x, since H does. This should not be a practical problem when training denoising auto-encoders.

Second, this choice only optimizes a Taylor approximation of the actual bound in Corollary 3, so it is only approximately optimal; see [GCB97]. Still, Corollary 3 applies to any choice of Σ so it provides a valid, exact bound for this approximately optimal choice.

Third, computing the Hessian $H(x)$ may not be practical. Still, since again Corollary 3 applies to an arbitrary Σ, it is not necessary to compute $H(x)$ exactly, and any reasonable approximation of $H(x)^{-1}$ yields a valid near-optimal bound and should provide a suitable order of magnitude for feature noise. [LBOM96, Section 7] provides useful Hessian approximations for neural networks, in particular the diagonal Gauss–Newton approximation (see the Appendix for more details).

In practice there are two different ways of using this result:

- One can use the denoising criterion of Corollary 3, in which at each step the noise level is set to an approximation of $H(x)^{-1}$, such as diagonal Gauss–Newton. This alternates between optimizing the model parameters for a given noise level, and optimizing the noise level for given model parameters.

- One can work directly with the objective function (28) from Theorem 5, which has an error term L_{rec} and a regularization term $\log \det H(x)$. Computing a gradient of the latter may be tricky. For multilayer neural networks, we provide in the Appendix (Theorem 9) an algorithm to compute this gradient at a cost of two forward and backpropagation passes if the layer-wise diagonal Gauss–Newton approximation of [LBOM96] is used for H. The algorithm is inspired from dynamic programming and the forward-backward algorithm used in hidden Markov models.

Proof of Lemma 4.
Using $y \sim \mathcal{N}(f(x), \Sigma)$ in Proposition 2, the reconstruction error $\mathbb{E}L_{\text{rec}}(x)$ is $\mathbb{E}_y L^y_{\text{rec}}(x)$. Using a second-order Taylor expansion of $L^y_{\text{rec}}(x)$ around $y = f(x)$, and using that $\mathbb{E}_{z\sim\mathcal{N}(0,\Sigma)}(z^\top M z) = \operatorname{Tr}(\Sigma M)$ for any matrix M, we find $\mathbb{E}L_{\text{rec}}(x) \approx L_{\text{rec}}(x) + \frac{1}{2}\operatorname{Tr}(\Sigma H_g)$ where H_g is the Hessian of $L^y_{\text{rec}}(x)$ at $y = f(x)$. By a similar argument the term $\operatorname{KL}(\mathcal{N}(f(x), \Sigma) \,\|\, \rho)$ is approximately $-\operatorname{Ent}\mathcal{N}(f(x), \Sigma) - \log\rho(f(x)) + \frac{1}{2}\operatorname{Tr}(\Sigma H_\rho)$ with H_ρ the Hessian of $-\log\rho(y)$ at $y = f(x)$. Thus the bound $L_{f,\Sigma\text{-gen}}(x)$ is approximately $L_{\text{rec}}(x) - \log\rho(f(x)) - \operatorname{Ent}\mathcal{N}(f(x), \Sigma) + \frac{1}{2}\operatorname{Tr}(\Sigma H)$ with $H = H_g + H_\rho$. The result follows from $\operatorname{Ent}\mathcal{N}(f(x), \Sigma) = \frac{1}{2}\log\det\Sigma + \frac{d}{2}(1 + \log 2\pi)$. □

Proof of Theorem 5.
Substituting $\Sigma = H(x)^{-1}$ in (25) directly yields the estimate in the proposition. Let us prove that this choice is optimal. We have to minimize $-\log\det\Sigma + \operatorname{Tr}(\Sigma H)$ over Σ. The case of diagonal Σ follows by direct minimization over the diagonal entries. For the general case, we have $\operatorname{Tr}(\Sigma H) = \operatorname{Tr}(H^{1/2}\Sigma H^{1/2})$. Since $H^{1/2}\Sigma H^{1/2}$ is symmetric we can decompose $H^{1/2}\Sigma H^{1/2} = O^\top D O$ with O orthogonal and D diagonal. Then $\operatorname{Tr}(\Sigma H) = \operatorname{Tr}(O^\top D O) = \operatorname{Tr}(D)$. Moreover, $\log\det\Sigma = \log\det(H^{-1/2}O^\top D O H^{-1/2}) = -\log\det H + \log\det D$ so that $-\log\det\Sigma + \operatorname{Tr}(\Sigma H) = \log\det H - \log\det D + \operatorname{Tr}(D) = \log\det H + \sum_k (d_k - \log d_k)$ with d_k the entries of D. The function $z \mapsto z - \log z$ is convex on $\mathbb{R}_+$ with a unique minimum at $z = 1$, so this is minimal if and only if $D = \mathrm{Id}$, i.e., $\Sigma = H^{-1}$. □

7 Link with contractive auto-encoders. The Hessian of the reconstruction error may not be easy to compute in practice. However, when reconstruction error is small this Hessian is related to the square derivatives of the reconstructed output with respect to the features, using the well-known Gauss–Newton approximation.

The resulting bound on codelength penalizes large square derivatives of the reconstructed outputs, as follows.

This is reminiscent of contractive auto-encoders ([RVM+11]; see also [Bis95] for the relationship between denoising and contractivity as regularization methods), with two differences: the contractivity is from features to output instead of from input to features, and instead of the Frobenius norm of the Jacobian matrix [RVM+11], the penalty is the sum of the logs of the norms of the rows of this matrix.

Proposition 6 (Codelength and contractivity). *Consider a quadratic reconstruction error of the type* $L = \sum_k \frac{(\hat{x}^k - x^k)^2}{2\sigma_k^2}$ *where* $\hat{x}^k$ *are the components of the reconstructed data* $\hat{x} = \hat{x}(y)$ *using features* y. *Let the elementary model* ρ *on* Y *be Gaussian with variance* $\operatorname{diag}(\lambda_i)$.

Then, when the reconstruction error is small enough,

$$L_{\text{rec}}(x) - \log \rho(f(x)) + \sum_i \log \sqrt{\frac{1}{\lambda_i} + \sum_k \frac{1}{\sigma_k^2} \left(\frac{\partial \hat{x}^k}{\partial y^i}\right)^2} - \frac{d}{2} \log 2\pi \tag{29}$$

is an approximate upper bound on $L_{\text{gen}}(x)$.

This corresponds to the approximately optimal choice $\Sigma = (\operatorname{diag} H)^{-1}$ together with the Gauss–Newton approximation $\frac{\partial^2 L}{\partial y^i \partial y^j} \approx \sum_k \frac{1}{\sigma_k^2} \frac{\partial \hat{x}^k}{\partial y^i} \frac{\partial \hat{x}^k}{\partial y^j}$.

The terms $1/\lambda_i$ prevent the logarithms from diverging to $-\infty$ in case a feature component i has no influence on the output $\hat{x}$. Typically λ_i will be large so the Jacobian norm $\sum_k \frac{1}{\sigma_k^2} \left(\frac{\partial \hat{x}^k}{\partial y^i}\right)^2$ dominates.

[RVM+11] contains an indication on how to optimize objective functions involving such derivatives for the case of a *single-layer* neural network: in that case the square derivatives are related to the squared weights of the network, so that the gradient of this term can be computed. For more complex models, however, $\partial \hat{x}^k / \partial y^i$ is a complex (though computable) function of the model parameters. Computing the gradient of $\left(\frac{\partial \hat{x}^k}{\partial y^i}\right)^2$ with respect to the model parameters is thus feasible but costly. Lemma 10 in the Appendix allows to compute a similar quantity for multilayer networks if the Gauss–Newton approximation is used on each layer in turn, instead of once globally from the y layer to the $\tilde{x}$ layer as used here. Optimizing (29) for multilayer networks using Lemma 10 would require $(\dim X)$ distinct backpropagations. More work is needed on this, such as stacking auto-encoders [HS06] to work with only one layer at a time.

Proof of Proposition 6.
Starting from Theorem 5, we have to approximate the Hessian $H(x) = \frac{\partial^2}{\partial y^2}(L^y_{\text{rec}}(x) - \log \rho(y))$. By the assumption that $L^y_{\text{rec}}(x) = \sum_k \frac{(\hat{x}^k - x^k)^2}{2\sigma_k^2}$, where $\hat{x}$ is a function of y, and since ρ is Gaussian, we get

$$H_{ij}(x) = \operatorname{diag}(1/\lambda_i) + \sum_k \frac{1}{2\sigma_k^2} \frac{\partial^2}{\partial y^i \partial y^j} (\hat{x}^k(y) - x^k)^2 \tag{30}$$

and we can use the well-known Gauss–Newton approximation [Bis06, 5.4.2], namely

$$\frac{\partial^2}{\partial y^i \partial y^j} (\hat{x}^k(y) - x^k)^2 = 2 \frac{\partial \hat{x}^k}{\partial y^i} \frac{\partial \hat{x}^k}{\partial y^j} + 2 \left(\hat{x}^k(y) - x^k\right) \frac{\partial^2 \hat{x}^k}{\partial y^i \partial y^j} \tag{31}$$

$$\approx 2 \frac{\partial \hat{x}^k}{\partial y^i} \frac{\partial \hat{x}^k}{\partial y^j} \tag{32}$$

valid whenever the error $\hat{x}^k(y) - x^k$ is small enough. (Interestingly, when summing over the dataset, it is not necessary that *every* error is small enough,

because errors with opposite signs will compensate; a fact used implicitly in [Bis95].)

The diagonal terms of $H(x)$ are thus

$$H_{ii}(x) \approx \frac{1}{\lambda_i} + \sum_k \frac{1}{\sigma_k^2} \left(\frac{\partial \hat{x}^k}{\partial y^i} \right)^2 \tag{33}$$

Now, from Theorem 5 the choice $\Sigma = (\operatorname{diag} H)^{-1}$ is optimal among diagonal noise matrices Σ. Computing the term $\frac{1}{2} \log \det \operatorname{diag} H = \sum_i \log \sqrt{H_{ii}}$ from (28) and substituting H_{ii} ends the proof. □

Remark 7. A tighter (approximately optimal) but less convenient bound is

$$L_{\text{rec}}(x) - \log \rho(f(x)) + \frac{1}{2} \log \det \left(-\frac{\partial^2 \log \rho(y)}{\partial y^i \partial y^j} + \sum_k \frac{1}{\sigma_k^2} \frac{\partial \hat{x}^k}{\partial y^i} \frac{\partial \hat{x}^k}{\partial y^j} \right)_{ij} - \frac{d}{2} \log 2\pi \tag{34}$$

which forgoes the diagonal approximation. For more general loss functions, a similar argument applies, resulting in a more complex expression which involves the Hessian of the loss with respect to the reconstruction $\hat{x}$.

Remark 8 (Adapting the elementary feature model ρ). Since all our bounds on L_{gen} involve $\log \rho(f(x))$ terms, the best choice of elementary model ρ is the one which maximizes the log-likelihood of the empirical feature distribution in space Y. This can be done concurrently with the optimization of the codelength, by re-adapting the prior after each step in the optimization of the functions f and g. For Gaussian models ρ as in Proposition 6, this leads to

$$\lambda_i \leftarrow \operatorname{Var}\left[f(x)^i \right] \tag{35}$$

with $f(x)^i$ the i-th component of feature $f(x)$, and x ranging over the dataset. If using the "denoising" criterion from Corollary 3, the noise on $f(x)$ must be included when computing this variance.

8 Variance of the output, and relative versus absolute error. A final, important choice when considering auto-encoders from a compression perspective is whether or not to include the variance of the output as a model parameter. While minimizing the reconstruction error usually focuses on absolute error, dividing the error by two will reduce codelength by one bit whether the error is large or small. This works out as follows.

Consider a situation where the outputs are real-valued (e.g., image). The usual loss is the square loss $L = \sum_n \sum_i (x_n^i - \hat{x}_n^i)^2$ where n goes through all samples and i goes through the components of each sample (output

dimension), the x_n are the actual data, and the $\hat{x}_n$ are the reconstructed data computed from the features y.

This square loss is recovered as the log-likelihood of the data over a Gaussian model with fixed variance σ and mean $\hat{x}_n$:

$$L_{\text{rec}}(\mathcal{D}) = -\sum_{x\in\mathcal{D}} \log g_{\hat{x}}(x) = \sum_{x\in\mathcal{D}}\sum_i \left(\frac{(x^i-\hat{x}^i)^2}{2\sigma^2} + \log\sigma + \frac{1}{2}\log 2\pi\right) \quad (36)$$

For any fixed σ, the optimum is the same as for the square loss above.

Incorporating a new parameter σ_i for the variance of the i-th component into the model may make a difference if the various output components have different scales or noise levels. The reconstruction error becomes

$$L_{\text{rec}}(\mathcal{D}) = \sum_i \sum_{x\in\mathcal{D}} \left(\frac{(x^i-\hat{x}^i)^2}{2\sigma_i^2} + \log\sigma_i + \frac{1}{2}\log 2\pi\right) \quad (37)$$

which is now to be optimized jointly over the functions f and g, and the σ_i's. The optimal σ_i for a given f and g is the mean square error[5] of component i,

$$\sigma_i^{*2} = E_i := \frac{1}{\#\mathcal{D}} \sum_{x\in\mathcal{D}} (x^i-\hat{x}^i)^2 \quad (38)$$

so with this optimal choice the reconstruction error is

$$L_{\text{rec}}(\mathcal{D}) = (\#\mathcal{D}) \sum_i \left(\frac{1}{2} + \frac{1}{2}\log E_i + \frac{1}{2}\log 2\pi\right) \quad (39)$$

and so we have to optimize

$$L_{\text{rec}}(\mathcal{D}) = \frac{\#\mathcal{D}}{2} \sum_i \log E_i + \text{Cst} \quad (40)$$

that is, the sum of the *logarithms* of the mean square error for each component. (Note that this is not additive over the dataset: each E_i is an average over the dataset.) Usually, the sum of the E_i themselves is used. Thus, including the σ_i as parameters changes the minimization problem by focusing on *relative* error, both for codelength and reconstruction error.

This is not cancelled out by normalizing the data: indeed the above does not depend on the variance of each component, but on the mean square prediction error, which can vary even if all components have the same variance, if some components are harder to predict.

This is to be used with caution when some errors become close to 0 (the log tends to $-\infty$). Indeed, optimizing this objective function means that

[5] If working with feature noise as in Corollary 3, this is the error after adding the noise. Optimizing σ_i for the estimate in Proposition 6 is more complicated since σ_i influences both the reconstruction error and the regularization term.

being able to predict an output component with an accuracy of 100 digits (for every sample x in the data) can balance out 100 bad predictions on other output components. This is only relevant if the data are actually precise up to 100 significant digits. In practice an error of 0 only means that the actual error is below the quantization level ε. Thus, numerically, we might want to consider that the smallest possible square error is ε^2, and to optimize $\sum_i \log(E_i + \varepsilon^2)$ for data quantized up to ε.

When working with the results of the previous sections (Prop. 2, Corollary 3, Thm. 5, and Prop. 6), changing σ has an influence: it changes the relative scaling of the reconstruction error term L_{rec} w.r.t. the remaining information-theoretic terms. Choosing the optimal σ_i as described here fixes this problem and makes all terms homogeneous.

Intuitively, from the minimum description length or compression viewpoint, dividing an error by 2 is an equally good move whether the error is small or large (one bit per sample gained on the codelength). Still, in a specific application, the relevant loss function may be the actual sum of square errors as usual, or a user-defined perceptual error. But in order to find a good representation of the data as an intermediate step in a final, user-defined problem, the compression point of view might be preferred.

Conclusions and perspectives

We have established that there is a strong relationship between minimizing a codelength of the data and minimizing reconstruction error using an auto-encoder. A variational approach provides a bound on data codelength in terms of the reconstruction error to which certain regularization terms are added.

The additional terms in the codelength bounds can be interpreted as a denoising condition from features to reconstructed output. This is in contrast with previously proposed denoising auto-encoders. For neural networks, this criterion can be trained using standard backpropagation techniques.

The codelength approach determines an optimal noise level for this denoising interpretation, namely, the one that will provide the tightest codelength. This optimal noise is approximately the inverse Hessian of the reconstruction function, for which several approximation techniques exist in the literature.

A practical consequence is that the noise level should be set differently for each data sample in a denoising approach.

Under certain approximations, the codelength approach also translates as a penalty for large derivatives from feature to output, different from that posited in contractive auto-encoders. However, the resulting criterion is hard to train for complex models such as multilayer neural networks. More work is needed on this point.

Including the variances of the outputs as parameters results in better compression bounds and a modified reconstruction error involving the *logarithms* of the square errors together with the data quantization level. Still, having these variances as parameters is a modeling choice that may be relevant for compression but not in applications where the actual reconstruction error is considered.

It would be interesting to explore the practical consequences of these insights. Another point in need of further inquiry is how this codelength viewpoint combines with the stacking approach to deep learning, namely, after the data x have been learned using features y and an elementary model for y, to further learn a finer model of y. For instance, it is likely that there is an interplay, in the denoising interpretation, between the noise level used on y when computing the codelength of x, and the output variance σ_y used in the definition of the reconstruction error of a model of y at the next level. This would require modeling the transmission of noise from one layer to another in stacked generative models and optimizing the levels of noise to minimize a resulting bound on codelength of the output.

References

[AO12] Ludovic Arnold and Yann Ollivier. Layer-wise learning of deep generative models. Preprint, arXiv:1212.1524, 2012.

[Bis95] Christopher M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995.

[Bis06] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[GCB97] Yves Grandvalet, Stéphane Canu, and Stéphane Boucheron. Noise injection: Theoretical prospects. *Neural Computation*, 9(5):1093–1108, 1997.

[Gra11] Alex Graves. Practical variational inference for neural networks. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 2348–2356, 2011.

[Grü07] Peter D. Grünwald. *The minimum description length principle*. MIT Press, 2007.

[HOT06] G.E. Hinton, S. Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[HS06] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

[HvC93] Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In Lenny Pitt, editor, *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, COLT 1993, Santa Cruz, CA, USA, July 26-28, 1993.*, pages 5–13. ACM, 1993.

[KW13] Diederik P. Kingma and Max Welling. Stochastic gradient VB and the variational auto-encoder. Preprint, arXiv:1312.6114, 2013.

[LBOM96] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 9–50. Springer, 1996.

[Oll13] Yann Ollivier. Riemannian metrics for neural networks I: feedforward networks. Preprint, http://arxiv.org/abs/1303.0818 , 2013.

[PH87] David C. Plaut and Geoffrey Hinton. Learning sets of filters using back-propagation. *Computer Speech and Language*, 2:35–61, 1987.

[RVM+11] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 833–840. Omnipress, 2011.

[VLL+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

Appendix: Derivative of $\log\det H$ for multilayer neural networks

The codelength bound from Theorem 5 involves a term $\log\det H(x)$ where $H(x)$ is the Hessian of the loss function for input x. Optimizing this term with respect to the model parameters is difficult in general.

We consider the case when the generative model $g\colon Y \to X$ is a multilayer neural network. We provide an algorithm to compute the derivative of the $\log\det H(x)$ term appearing in Theorem 5 with respect to the network weights, using the layer-wise diagonal Gauss–Newton approximation of the Hessian $H(x)$ from [LBOM96]. The algorithm has the same asymptotic computational cost as backpropagation.

So let the generative model g be a multilayer neural network with activation function s. The activity of unit i is

$$a_i := s(V_i), \qquad V_i := \sum_{j\to i} a_j w_{ji} \tag{41}$$

where the sum includes the bias term via the always-activated unit $j = 0$ with $a_j \equiv 1$.

Let L be the loss function of the network.

The layer-wise diagonal Gauss–Newton approximation computes an approximation $\mathfrak{h}_i$ to the Hessian $\frac{\partial^2 L}{\partial a_i^2}$ in the following way [LBOM96, Sections 7.3–7.4]: On the output units k, $\mathfrak{h}_k$ is directly set to $\mathfrak{h}_k := \frac{\partial^2 L}{\partial a_k^2}$, and this is backpropagated through the network via

$$\mathfrak{h}_i := \sum_{j,\, i\to j} \left(\frac{\partial a_j}{\partial a_i}\right)^2 \mathfrak{h}_j = \sum_{j,\, i\to j} w_{ij}^2\, s'(V_j)^2\, \mathfrak{h}_j \tag{42}$$

so that computing $\mathfrak{h}_i$ is similar to backpropagation using squared weights. This is also related to the *backpropagated metric* from [Oll13].

Theorem 9 (Gradient of the determinant of the Gauss–Newton Hessian). *Consider a generative model g given by a multilayer neural network. Let the reconstruction error be $L = \sum_k \frac{(\hat{x}^k - x^k)^2}{2\sigma_k^2}$ where $\hat{x}^k$ are the components of the reconstructed data $\hat{x} = \hat{x}(y)$ using features y. Let the elementary model ρ on Y be Gaussian with variance $\operatorname{diag}(\lambda_i)$.*

Let $H(x) = \frac{\partial^2}{\partial y^2}(L^y_{\mathrm{rec}}(x) - \log\rho(y))$ as in Theorem 5. Let $\hat{H}(x)$ be the layer-wise diagonal Gauss–Newton approximation of $H(x)$, namely

$$\hat{H}(x) := \operatorname{diag}\left(\lambda_i^{-1} + \mathfrak{h}_i\right) \tag{43}$$

with $\mathfrak{h}_i$ computed from (42), initialized via $\mathfrak{h}_k = 1/\sigma_k^2$ on the output layer.

Then the derivative of $\log\det\hat{H}(x)$ with respect to the network weights w can be computed exactly with an algorithmic cost of two forward and backpropagation passes.

This computation is trickier than it looks because the coefficients $s'(V_j)^2$ used in the backpropagation for $\mathfrak{h}$ depend on the weights of all units before j (because V_j does), not only the units directly influencing j.

Proof.
Apply the following lemma with $\mathfrak{B} = \mathfrak{h}$, $\varphi(w, V) = w^2 s'(V)^2$, and $\psi_i(\mathfrak{h}_i) = \log(\lambda_i^{-1} + \mathfrak{h}_i)$. □

Lemma 10 (Gradients of backpropagated quantities). *Let $\mathfrak{B}$ be a function of the state of a neural network computed according to the backpropagation equation*

$$\mathfrak{B}_i = \sum_{j,\, i \to j} \varphi_j(w_{ij}, V_j) \mathfrak{B}_j \tag{44}$$

initialized with some fixed values $\mathfrak{B}_k$ on the output layer.

Let

$$S := \sum_{i \in \mathcal{L}_{\text{in}}} \psi_i(\mathfrak{B}_i) \tag{45}$$

for some functions ψ_i on the input layer $\mathcal{L}_{\text{in}}$.

Then the derivatives of S with respect to the network parameters w_{ij} can be computed at the same algorithmic cost as one forward and two backpropagation passes, as follows.

1. *Compute $\mathfrak{B}_i$ for all i by backpropagation.*
2. *Compute the variable $\mathfrak{C}_j$ by forward propagation for all units j, as*

$$\mathfrak{C}_j := \sum_{i \to j} \mathfrak{C}_i \varphi_j(w_{ij}, V_j) \tag{46}$$

   initialized with $\mathfrak{C}_i = \psi_i'(\mathfrak{B}_i)$ for i in the input layer.

3. *Compute the variable D_i by backpropagation for all units i, as*

$$D_i := \sum_{k,\, k \to i} \mathfrak{C}_k \mathfrak{B}_i \frac{\partial \varphi_i(w_{ki}, V_i)}{\partial V_i} + \sum_{j,\, i \to j} s'(V_i) w_{ij} D_j \tag{47}$$

   (also used for initialization with i in the output layer, with an empty sum in the second term).

Then the derivatives of S are

$$\frac{\partial S}{\partial w_{ij}} = \mathfrak{C}_i \mathfrak{B}_j \frac{\partial \varphi_j(w_{ij}, V_j)}{\partial w_{ij}} + a_i D_j \tag{48}$$

for all i, j.

Note that we assume that the values $\mathfrak{B}_k$ used to initialize $\mathfrak{B}$ on the output layer are fixed (do not depend on the network weights). Any dependency of $\mathfrak{B}_k$ on the output layer activity values a_k can, instead, be incorporated into φ_k via V_k.

Proof.
We assume that the network is an arbitrary finite, directed acyclic graph. We also assume (for simplicity only) that no unit is both an output unit and influences other units. We denote $i \to j$ if there is an edge from i to j, $i > j$ if there is a path of length $\geqslant 1$ from i to j, and $i \geqslant j$ if $i > j$ or $i = j$.

The computation has a structure similar to the forward-backward algorithm used in hidden Markov models.

For any pair of units l, m in the network, define the "backpropagation transfer rate" [Oll13] from l to m as

$$\tau_l^m := \sum_\gamma \prod_{t=1}^{|\gamma|} \varphi_{\gamma_t}(w_{\gamma_{t-1}\gamma_t}, V_{\gamma_t}) \tag{49}$$

where the sum is over all paths γ from l to m in the network (including the length-0 path for $l = m$), and $|\gamma|$ is the length of γ. In particular, $\tau_m^m = 1$ and $\tau_l^m = 0$ if there is no path from l to m. By construction these satisfy the backpropagation equation

$$\tau_i^k = \sum_{j,\, i\to j} \varphi_j(w_{ij}, V_j)\tau_j^k \tag{50}$$

for $i \neq k$. By induction

$$\mathfrak{B}_i = \sum_{k\in\mathcal{L}_{\text{out}}} \tau_i^k \mathfrak{B}_k \tag{51}$$

where the sum is over k in the output layer $\mathcal{L}_{\text{out}}$. Consequently the derivative of $S = \sum_{i\in\mathcal{L}_{\text{in}}} \psi_i(\mathfrak{B}_i)$ with respect to a weight w_{mn} is

$$\frac{\partial S}{\partial w_{mn}} = \sum_{i\in\mathcal{L}_{\text{in}}} \psi_i'(\mathfrak{B}_i) \sum_{k\in\mathcal{L}_{\text{out}}} \frac{\partial \tau_i^k}{\partial w_{mn}} \mathfrak{B}_k \tag{52}$$

so that we have to compute the derivatives of τ_i^k. (This assumes that the initialization of $\mathfrak{B}_k$ on the output layer does not depend on the weights w.)

A weight w_{mn} influences $\varphi_n(w_{mn}, V_n)$ and also influences V_n which in turn influences all values of V_j at subsequent units. Let us first compute the derivative of τ_i^k with respect to V_n. Summing over paths γ from i to k we find

$$\frac{\partial \tau_i^k}{\partial V_n} = \sum_\gamma \frac{\partial}{\partial V_n} \prod_{t=1}^{|\gamma|} \varphi_{\gamma_t}(w_{\gamma_{t-1}\gamma_t}, V_{\gamma_t}) \tag{53}$$

$$= \sum_\gamma \sum_t \left(\prod_{s=1}^{t-1} \varphi_{\gamma_s}(w_{\gamma_{s-1}\gamma_s}, V_{\gamma_s}) \right) \frac{\partial \varphi_{\gamma_t}(w_{\gamma_{t-1}\gamma_t}, V_{\gamma_t})}{\partial V_n} \left(\prod_{s=t}^{|\gamma|} \varphi_{\gamma_s}(w_{\gamma_{s-1}\gamma_s}, V_{\gamma_s}) \right) \tag{54}$$

$$= \sum_{(l,m),\, l\to m} \tau_i^l \frac{\partial \varphi_m(w_{lm}, V_m)}{\partial V_n} \tau_m^k \tag{55}$$

by substituting $l = \gamma_{t-1}$, $m = \gamma_t$ for each value of t, and unraveling the definition of τ_i^l and τ_m^k.

Since V_n only influences later units in the network, the only non-zero terms are those with $n \geqslant m$. We can decompose into $m = n$ and $n > m$:

$$\frac{\partial \tau_i^k}{\partial V_n} = \sum_{l,\, l\to n} \tau_i^l \frac{\partial \varphi_n(w_{ln}, V_n)}{\partial V_n} \tau_n^k + \sum_{m,n>m} \sum_{l,\, l\to m} \tau_i^l \frac{\partial \varphi_m(w_{lm}, V_m)}{\partial V_n} \tau_m^k \tag{56}$$

Now, for $n > m$, the influence of V_n on V_m has to transit through some unit j directly connected to n, namely, for any function $\mathcal{F}(V_m)$,

$$\frac{\partial \mathcal{F}(V_m)}{\partial V_n} = \sum_{j,\, n\to j} s'(V_n) w_{nj} \frac{\partial \mathcal{F}(V_m)}{\partial V_j} \tag{57}$$

where s is the activation function of the network. So

$$\sum_{m,\, n>m} \sum_{l,\, l\to m} \tau_i^l \frac{\partial \varphi_m(w_{lm}, V_m)}{\partial V_n} \tau_m^k = \sum_{j,\, n\to j} s'(V_n) w_{nj} \sum_{m,\, n>m} \sum_{l,\, l\to m} \tau_i^l \frac{\partial \varphi_m(w_{lm}, V_m)}{\partial V_j} \tau_m^k \tag{58}$$

$$= \sum_{j,\, n\to j} s'(V_n) w_{nj} \sum_{m} \sum_{l,\, l\to m} \tau_i^l \frac{\partial \varphi_m(w_{lm}, V_m)}{\partial V_j} \tau_m^k \tag{59}$$

where the difference between the last two lines is that we removed the condition $n > m$ in the summation over m: indeed, any m with non-vanishing $\partial V_m / \partial V_j$ satisfies $j \geqslant m$ hence $n > m$. According to (55), $\sum_m \sum_{l,\, l\to m} \tau_i^l \frac{\partial \varphi_m(w_{lm}, V_m)}{\partial V_j} \tau_m^k$ is $\frac{\partial \tau_i^k}{\partial V_j}$, so that (59) is $\sum_{j,\, n\to j} s'(V_n) w_{nj} \frac{\partial \tau_i^k}{\partial V_j}$. Collecting from (56), we find

$$\frac{\partial \tau_i^k}{\partial V_n} = \sum_{l,\, l\to n} \tau_i^l \frac{\partial \varphi_n(w_{ln}, V_n)}{\partial V_n} \tau_n^k + \sum_{j,\, n\to j} s'(V_n) w_{nj} \frac{\partial \tau_i^k}{\partial V_j} \tag{60}$$

so that the quantities $\frac{\partial \tau_i^k}{\partial V_n}$ can be computed by backpropagation on n, if the τ are known.

To compute the derivatives of τ_i^k with respect to a weight w_{mn}, observe that w_{mn} influences the w_{mn} term in $\varphi_n(w_{mn}, V_n)$, as well as all terms V_l with $n \geqslant l$ via its influence on V_n. Since $\frac{\partial V_n}{\partial w_{mn}} = a_m$ we find

$$\frac{\partial \varphi_l(w_{jl}, V_l)}{\partial w_{mn}} = \mathbb{1}_{(j,l)=(m,n)} \frac{\partial \varphi_n(w_{mn}, V_n)}{\partial w_{mn}} + a_m \frac{\partial \varphi_n(w_{jl}, V_l)}{\partial V_n} \tag{61}$$

By following the same procedure as in (53)–(55) we obtain

$$\frac{\partial \tau_i^k}{\partial w_{mn}} = \tau_i^m \frac{\partial \varphi_n(w_{mn}, V_n)}{\partial w_{mn}} \tau_n^k + a_m \sum_{(j,l),\, j\to l} \tau_i^j \frac{\partial \varphi_l(w_{jl}, V_l)}{\partial V_n} \tau_l^k \tag{62}$$

$$= \tau_i^m \frac{\partial \varphi_n(w_{mn}, V_n)}{\partial w_{mn}} \tau_n^k + a_m \frac{\partial \tau_i^k}{\partial V_n} \tag{63}$$

by (53).

This allows, in principle, to compute the desired derivatives. By (52) we have to compute the sum of (63) over $i \in \mathcal{L}_{\text{in}}$ and $k \in \mathcal{L}_{\text{out}}$ weighted by $\psi_i'(\mathfrak{B}_i)$ and $\mathfrak{B}_k$. This avoids a full computation of all transfer rates τ and yields

$$\frac{\partial S}{\partial w_{mn}} = \mathfrak{C}_m \frac{\partial \varphi_n(w_{mn}, V_n)}{\partial w_{mn}} \mathfrak{B}_n + a_m D_n \tag{64}$$

where we have set

$$\mathfrak{C}_m := \sum_{i \in \mathcal{L}_{\text{in}}} \psi_i'(\mathfrak{B}_i) \tau_i^m, \tag{65}$$

and

$$D_n := \sum_{i \in \mathcal{L}_{\text{in}}} \sum_{k \in \mathcal{L}_{\text{out}}} \psi_i'(\mathfrak{B}_i) \mathfrak{B}_k \frac{\partial \tau_i^k}{\partial V_n} \tag{66}$$

and where we have used that $\mathfrak{B}$ satisfies

$$\mathfrak{B}_m = \sum_{k \in \mathcal{L}_{\text{out}}} \tau_m^k \mathfrak{B}_k \tag{67}$$

by (51).

It remains to provide ways to compute $\mathfrak{C}_m$ and D_n. For $\mathfrak{C}_m$, note that the transfer rates τ satisfy the forward propagation equation

$$\tau_i^k = \sum_{j,\, j \to k} \varphi_k(w_{jk}, V_k) \tau_i^j \tag{68}$$

by construction. Summing over $i \in \mathcal{L}_{\text{in}}$ with weights $\psi_i'(\mathfrak{B}_i)$ yields the forward propagation equation for $\mathfrak{C}$ given in the statement of the lemma.

Finally, by summing over i and k in (60), with weights $\psi_i'(\mathfrak{B}_i)\mathfrak{B}_k$, and using the definition of $\mathfrak{C}$ and again the property $\mathfrak{B}_n = \sum_{k \in \mathcal{L}_{\text{out}}} \tau_n^k \mathfrak{B}_k$, we obtain

$$D_n = \sum_{l,\, l \to n} \mathfrak{C}_l \frac{\partial \varphi_n(w_{ln}, V_n)}{\partial V_n} \mathfrak{B}_n + \sum_{j,\, n \to j} s'(V_n) w_{nj} D_j \tag{69}$$

which is the backpropagation equation for D_n and concludes the proof. □