Autoencoders

Carlos Valle

Departamento de Informática
Universidad Técnica Federico Santa María

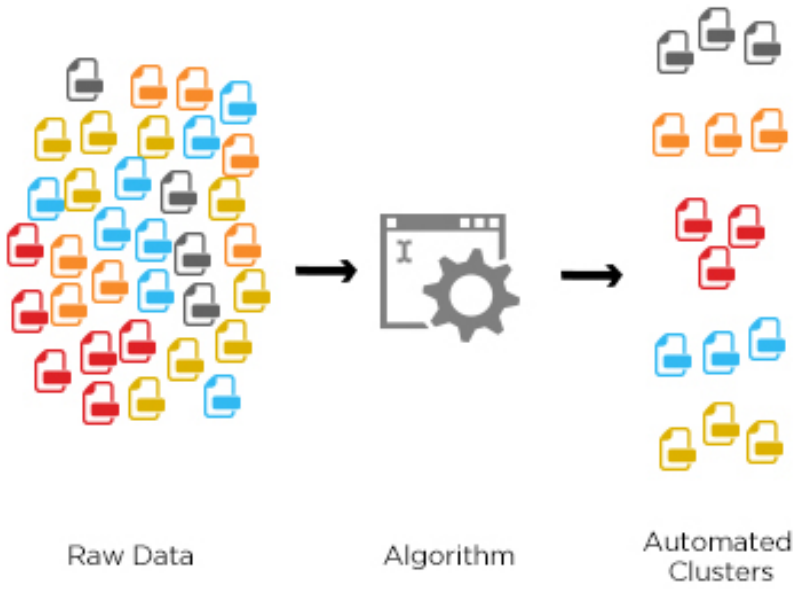
cvalle@inf.utfsm.cl

December 15, 2018

Overview

Supervised vs Unsupervised

- Supervised Learning: To Learn the map between inputs and outputs.
- Unsupervised Learning: Input data is not labeled. To Find structures deduced on inputs data.



Autoencoders

- An autoencoder is a neural network that is trained to attempt to copy its input to its output.
- Internally, it has a hidden layer h that describes a code used to represent the input.
- Self-supervised: A form of unsupervised learning where the data provides the supervision.

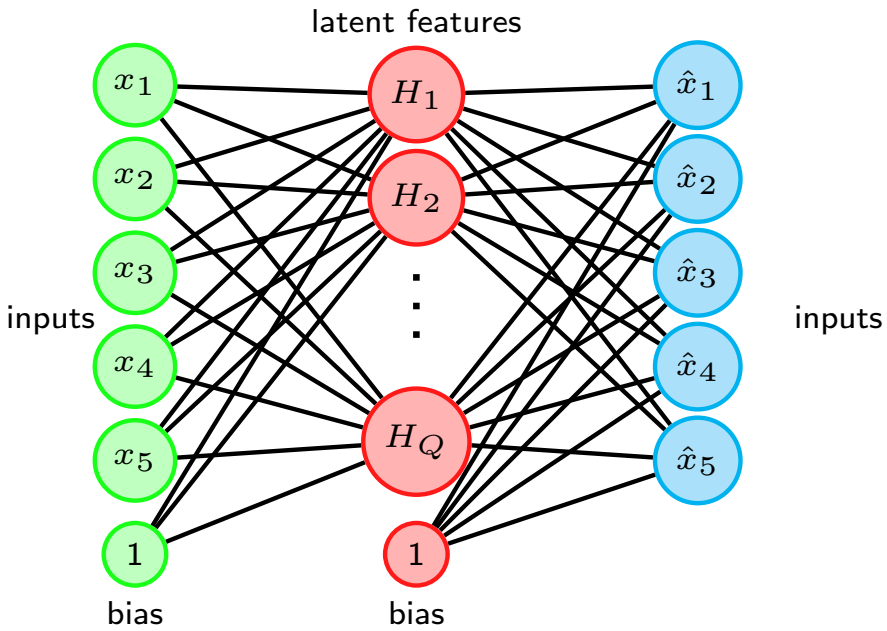
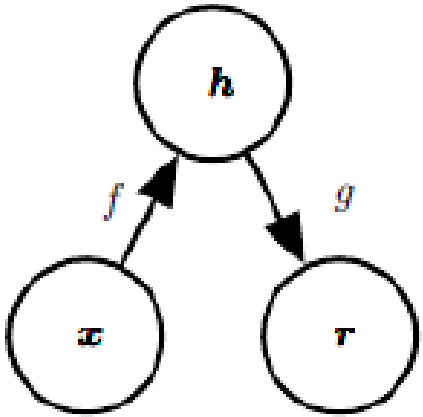


Figure 1: Autoencoder scheme

Autoencoders (2)

- The network may be viewed as consisting of two parts: an encoder function $h = f(\mathbf{x})$ and a decoder that produces a reconstruction $r = g(h)$.
- An autoencoder tries to learn $g(f(\mathbf{x})) = \mathbf{x}$. However, an autoencoder can be restricted to learn useful properties of data, instead of copying it.



Autoencoders (3)

- Modern autoencoders have generalized the idea of an encoder and a decoder beyond deterministic functions to stochastic mappings $p_{encoder}(h|\mathbf{x})$ and $p_{decoder}(\mathbf{x}|h)$.
- We train the autoencoder to minimize the loss function $\ell(\mathbf{x}, g(f(\mathbf{x})))$ typically the mean squared error.

Undercomplete autoencoders

- We hope that training the autoencoder to perform the input copying task will result in h taking on useful properties.
- One way to obtain useful features from the autoencoder is to constrain h to have smaller dimension than $\mathbf{x}$.
- An autoencoder whose code dimension is less than the input dimension is called *undercomplete* (less neurons).
- More neurons: *overcomplete* autoencoder.

Undercomplete autoencoders (2)

- Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data.
- When the decoder is linear and ℓ is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA.
- Autoencoders with nonlinear encoder functions f and nonlinear decoder functions g can thus learn a more powerful nonlinear generalization of PCA.
- However, if the encoder and decoder are allowed too much capacity, the autoencoder might perform the copying task without extracting useful information about the distribution of the data.

Regularized autoencoders

- Problem: How to choose the code dimension and the capacity of the encoder and decoder based on the complexity of distribution to be modeled?
- Regularized autoencoders provide the ability to do so. Rather than limiting the model capacity by keeping the encoder and decoder shallow and the code size small.
- Regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output.
- These other properties include:
  - Sparsity of the representation.
  - Robustness to noise or to missing inputs.
  - Smallness of the derivative of the representation.

Sparse autoencoders

- A sparse autoencoder adds a sparsity penalty $\Omega(h)$ on the code layer h, in addition to the reconstruction error:

$$\ell(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(h),$$

- where $g(h)$ is the decoder output and typically we have $h = f(\mathbf{x})$, the encoder output.
- Before studying sparse autoencoders, we will study the Maximum a Posteriori (MAP) point of view of neural networks.

Maximum A Posteriori (MAP)

- From Bayesian Inference and Maximum A Posteriori (MAP), neural networks can be written as:

$$P(w|D) = \frac{P(w)P(D|w)}{\int P(w)P(D|w)}.$$

- Where, $P(w|D)$ is the posterior probability of the weight vector w given the training data set D.
- $P(w)$ is the prior probability of the weight vector.
- $P(D|w)$ is the probability of the observed data given weight vector w.
- And, the denominator is the integral of all possible weight vectors.

Maximum A Posteriori (MAP) (2)

- We can convert the above equation to a cost function again applying the negative log likelihood as follows:

$$cost = -\log(P(w|D)),$$

$$\log P(w|D) = \log P(w) + \log P(D|w) - \log(P(D)).$$

- Here $P(D)$ is the integral over all possible weights and hence $\log P(D)$ is some constant.
- From the Maximum Likelihood, we already learnt the equation for $\log P(D|w)$.
- On the other hand, using maximum likelihood:

$$P(D|w) = \prod_m P(y_m|w) = \prod_m P(y_m|f(x_m, w)).$$

- Which yields

$$-\log P(D|w) = \sum_m -\log P(y_m|\hat{y}_m).$$

Maximum A Posteriori (MAP) (3)

- For example, if we assume Gaussian noise to the output of the network:

$$P(y_m|\hat{y}_m) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_m-\hat{y}_m)^2}{2\sigma^2}}$$

- Then,

$$-\log P(y_m|\hat{y}_m) = K + \frac{(y_m-\hat{y}_m)^2}{2\sigma_D^2}$$

- If we assume that the network weights are drawn from $N(0,\sigma_w^2)$:

$$\frac{1}{\sqrt{2\pi}\sigma_w} e^{-\frac{w^2}{2\sigma_w^2}}$$

- Thus, $-\log P(w) = \frac{-w^2}{2\sigma_w^2}$.

Maximum A Posteriori (MAP) (4)

- Now we have:

$$
\begin{aligned}
cost &= -\log P(w) - \log P(D|w) + \log(P(D)) \\
&= \frac{1}{2\sigma_w^2}\sum w_i^2 - \frac{1}{2\sigma_D^2}\sum_m (y_m - \hat{y}_m)^2 \\
&= \frac{1}{2\sigma_D^2}\left[\frac{\sigma_D^2}{\sigma_w^2}\sum w_i^2 + E\right] \\
&= \frac{1}{2\sigma_D^2}\left[\frac{\sigma_D^2}{\sigma_w^2}\sum w_i^2 + E\right] \\
&= \frac{\sigma_D^2}{\sigma_w^2}\sum w_i^2 + E
\end{aligned}
$$

- This is equivalent to train with weight decay using $\lambda = \frac{\sigma_D^2}{\sigma_w^2}$

Going back to sparse autoencoders

- Since the regularizer $\Omega(h)$ depends on the data, by definition this is not a prior for the MAP view.
- But, we can think of the entire sparse autoencoder framework as approximating maximum likelihood training of a generative model that has latent variables.
- Suppose we have a model with visible variables $\mathbf{x}$ and latent variables h, with an explicit joint distribution:
- $$p_{model}(\mathbf{x}, h) = p_{model}(h)p_{model}(\mathbf{x}|h).$$
- Here, $p_{model}(h)$ refers the model's prior distribution over the latent variables, representing the model' s beliefs prior to seeing $\mathbf{x}$.

Going back to sparse autoencoders (2)

- MLE:

$$\log p_{model}(x) = \log \sum_{h'} p(h', x)$$

- Suppose $h = \mathbf{x}$ gives the most likely hidden representation.
- Then, $\sum_{h'} p(h', x)$ can be approximated by $p(h, x)$
- Hence with this assumption h can be chosen maximizing

$$\log p_{model}(h, \mathbf{x}) = \log p_{model}(h) \log p_{model}(\mathbf{x}|h).$$

- The $\log p_{model}(h)$ term can be sparsity-inducing.
- For example, the Laplace prior,

$$p_{model}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|},$$

corresponds to an absolute value sparsity penalty. Expressing the log-prior as an absolute value penalty, we obtain

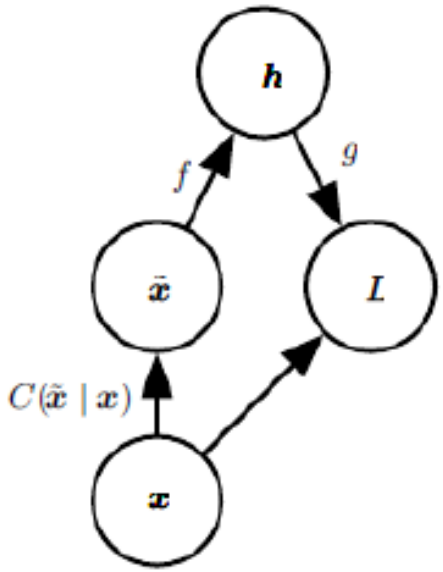
$$\Omega(h) = \lambda \sum_i |h_i|$$

Denoising autoencoders

- A denoising autoencoder or DAE instead minimizes

$$\ell(\mathbf{x}, g(f(\tilde{x})))$$

  where $\tilde{\mathbf{x}}$ is a copy of $\mathbf{x}$ that has been corrupted by some form of noise.
- Denoising autoencoders must learn the underlying structure of $\mathbf{x}$ than simply copying their input.
- $C(\tilde{\mathbf{x}}|\mathbf{x})$ represents a conditional distribution over corrupted samples $\tilde{\mathbf{x}}$ given a data sample $\mathbf{x}$.



Denoising autoencoders (2)

- So long as the encoder is deterministic, the denoising autoencoder is a feedforward that minimizes

$$\ell = -\log p_{decoder}(\mathbf{x}|h = f(\tilde{x}))$$

- We can therefore view the DAE as performing stochastic gradient descent on the following expectation:

$$-E_{\mathbf{x}\sim\tilde{p}_{data}(\mathbf{x})}E_{\tilde{\mathbf{x}}\sim C(\tilde{\mathbf{x}}|\mathbf{x})}\log p_{decoder}(\mathbf{x}|h = f(\tilde{\mathbf{x}}),)$$

- where $\tilde{p}_data$ is the training distribution.
- For instance, we can use additive Gaussian noise $\tilde{\mathbf{x}} \sim N(\mathbf{x}, \sigma^2 I)$,
- or salt-and-pepper noise, which sets a fraction of the elements of the input to their minimum or maximum value, according to a uniform distribution.

Contractive autoencoders

- Rifai et al, 2011. Contractive Auto-Encoders: Explicit Invariance During Feature Extraction.
- We need a regularizer Ω:

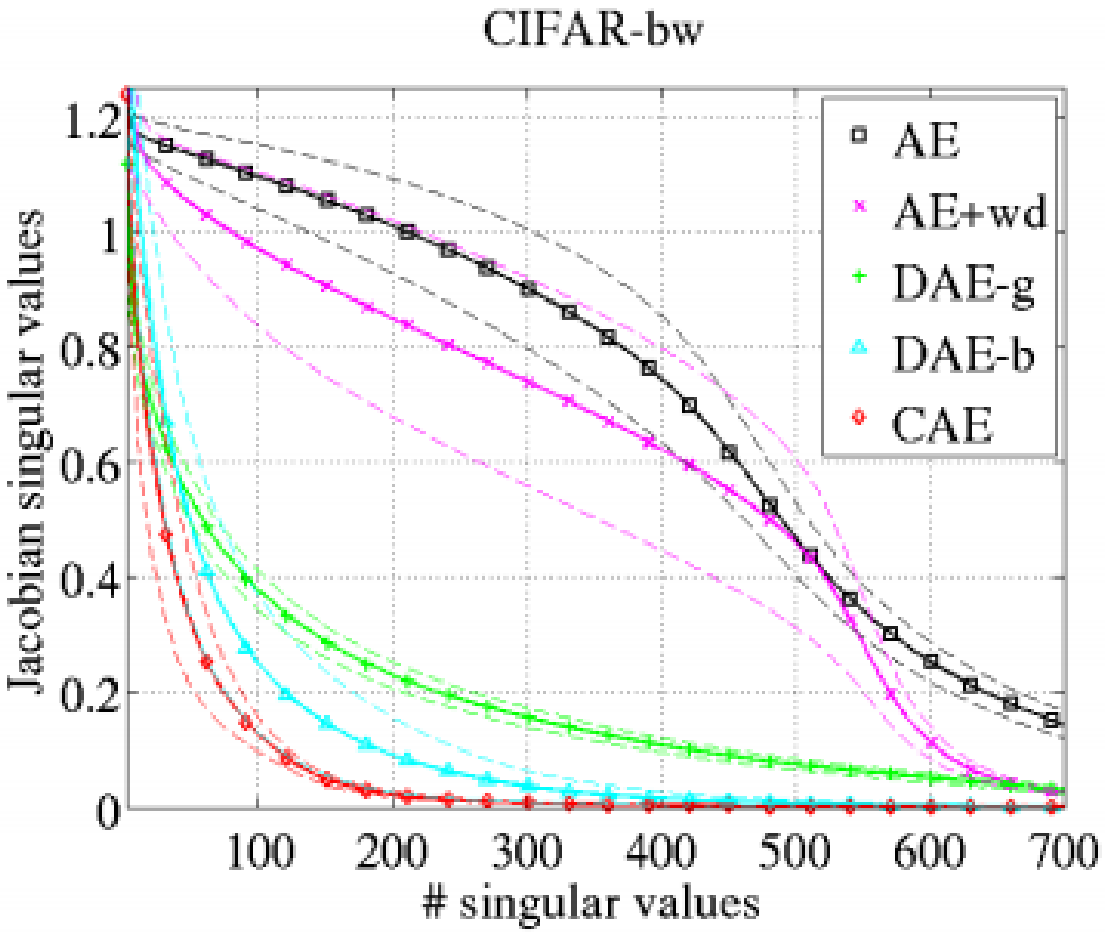
$$\Omega(h, \mathbf{x}) = \lambda \sum_i ||\nabla_x h_i||^2$$

- This forces the model to learn a function that does not change much when x changes slightly. Because this penalty is applied only at training examples, it forces the autoencoder to learn features that capture information about the training distribution.

Contractive autoencoders (2)

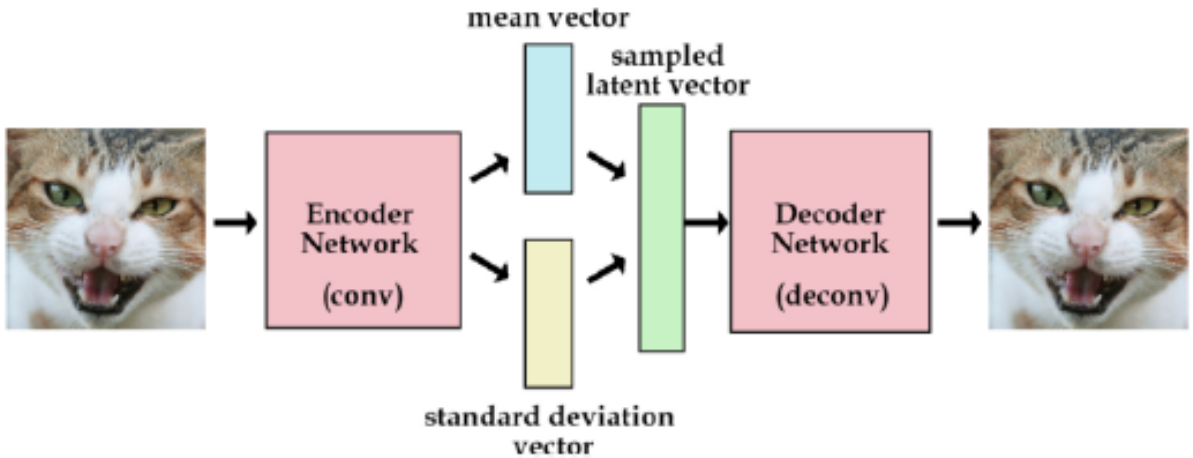
- The penalty $\Omega(h)$ is the squared Frobenius norm (sum of squared elements) of the Jacobian matrix of partial derivatives associated with the encoder function:
- $$\Omega(h) = \lambda \left\| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right\|_F^2$$
- The squared Frobenius norm of the Jacobian corresponds to a L2 weight decay in the case of a linear encode.
- This regularizer also encourages an sparse autoencoder.
- Authors empirically show that eigenvalues are less than one.

Contractive autoencoders (2)



Autoencoders as generative models

- Given an observable variable X and a target variable Y, a generative model is a statistical model of the joint probability distribution on $X \times Y$, $P(X, Y) = P(X|Y)P(Y)$.
- A discriminative model model the conditional probability of the target Y, given an observation $\mathbf{x}$, $P(Y|X = x)$.
- Classifiers computed without using a probability model are also referred to loosely as "discriminative".



Variational Autoencoders (VAEs)

- Variational autoencoders first draws a sample z from the code distribution $p_{model}(z)$. The sample is then run through a differentiable generator network $g(z)$.
- Finally, x is sampled from a distribution $p_{model}(x; g(z)) = p_{model}(x|z)$
- $p_{model}(x|z)$ is then viewed as a decoder network.
- However, during training, the approximate inference network (or encoder) $q(z|x)$ is used to obtain z.
- Approximation trick: z is set as $\mu + \sigma \circ N(0, 1)$

Variational Autoencoders (2)

- The key insight behind variational autoencoders is that they may be trained by maximizing the variational lower bound $\mathcal{L}(q)$ associated with data point $\mathbf{x}$:

$$\begin{aligned}\mathcal{L}(q) &= E_{z\sim q(z|x)} \log p_{model}(z,x) + H(q(z|x)) \\ &= E_{z\sim q(z|x)} \log p_{model}(x|z) - DKL(q(z|x)||pmodel(z)) \\ &\leq \log p_{model}(x).\end{aligned}$$

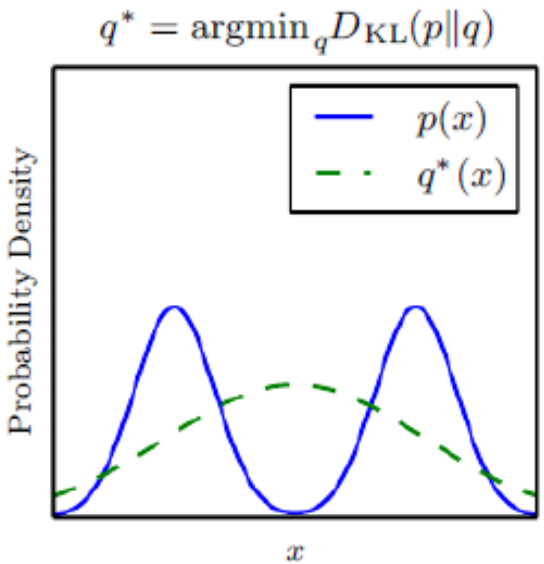
- where DKL is the Kullback-Leibler divergence defined as:

$$\text{DKL}(P\|Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx$$

- Thus, this lower bound is computed with two terms: First one is the reconstruction log-likelihood. The second one tries to make the approximate posterior distribution $q(z|x)$ and the model prior $p_model(z)$ approach each other.

Variational Autoencoders (3)

- When q is chosen to be a Gaussian distribution, with noise added to a predicted mean value, maximizing this entropy term encourages increasing the standard deviation of this noise.
- More generally, this entropy term encourages the variational posterior to place high probability mass on many z values that could have generated x, rather than collapsing to a single point estimate of the most likely value.



Deep autoencoders

- As we have discussed, deep neural networks have many advantages: deph can exponentially reduce the computational cost of representing some functions. Depth can also exponentially decrease the amount of training data needed to learn some functions.
- In particular a deep encoder is able to enforce arbitrary constraints such as sparse code.
- A common strategy for training a deep autoencoder is to greedily pretrain the deep architecture by training a stack of shallow autoencoders.

Any questions?