Lecture 20: AutoVC: Zero-Shot Voice Conversion using Autoencoder Loss

Mark Hasegawa-Johnson

University of Illinois

ECE 417: Multimedia Signal Processing

Outline

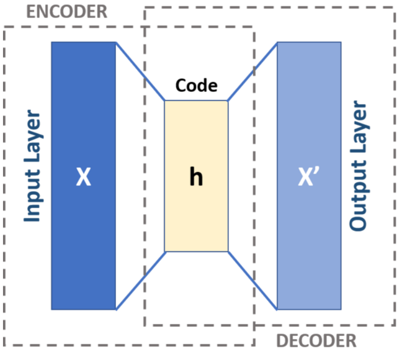
Autoencoder

A two-layer network is a network with two matrix multiplications, e.g.,

$$\mathbf{h} = g\left(\mathbf{W}_1\mathbf{x}\right)$$
$$\mathbf{x}' = \mathbf{W}_2\mathbf{h}$$

An **autoencoder** is a neural net trained to minimize the difference between its output and its input:

$$\mathcal{L} = \|\mathbf{x}' - \mathbf{x}\|^2$$



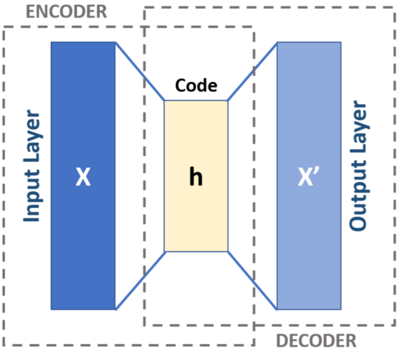
`https://commons.wikimedia.org/wiki/File:Autoencoder_schema.png`

Dimensionality Reduction

Notice that if len($\boldsymbol{h}$) = len($\boldsymbol{x}$), then there is a trivial solution:

$$\boldsymbol{h} = \boldsymbol{I}\boldsymbol{x} = \boldsymbol{x}$$
$$\boldsymbol{x}' = \boldsymbol{I}\boldsymbol{h} = \boldsymbol{x}$$
$$\mathcal{L} = \|\boldsymbol{x}' - \boldsymbol{x}\|^2 = 0$$

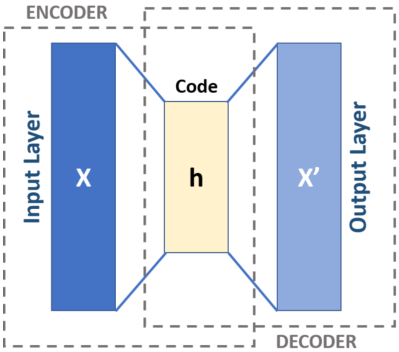
For this reason, an autoencoder is only interesting if $\boldsymbol{h}$ is limited in some way. Most often, it is because len($\boldsymbol{h}$) < len($\boldsymbol{x}$).



`https://commons.wikimedia.org/wiki/File:Autoencoder_schema.png`

Dimensionality reduction

The goal of the autoencoder is to learn a vector ***h*** that is shorter than ***x***, but contains most of the same information.



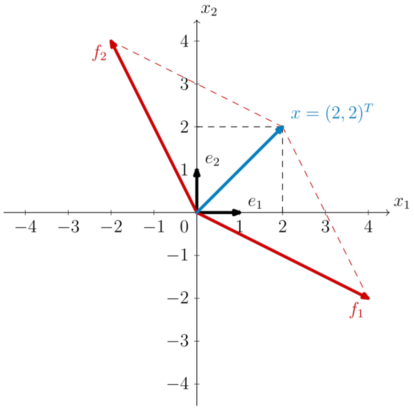
`https://commons.wikimedia.org/wiki/File:Autoencoder_schema.png`

Linear Autoencoder = Smart PCA

Suppose that the hidden layer is linear, $\text{len}(\boldsymbol{h}) = K$, $\text{len}(\boldsymbol{x}) = D$, and suppose that $\boldsymbol{W}_1 = \boldsymbol{W}_2^T = \boldsymbol{W}^T$, i.e.,

$$\boldsymbol{h} = g\left(\boldsymbol{W}^T\boldsymbol{x}\right)$$
$$\boldsymbol{x}' = \boldsymbol{W}\boldsymbol{h}$$
$$\mathcal{L} = \|\boldsymbol{x}' - \boldsymbol{x}\|^2$$

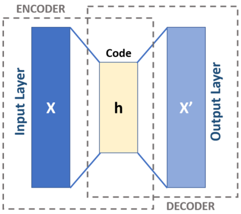
In order to minimize $\mathcal{L}$, the columns of $\boldsymbol{W}$ must span the same space as the first K principal components. They don't need to be orthogonal, but they must span the space.



https://commons.wikimedia.org/wiki/File:Change_of_basis_22.svg

Linear Autoencoder = Smart PCA

- If $\boldsymbol{x} \in \Re^D$, then regular PCA finds the eigenvectors of the covariance matrix: an $\mathcal{O}\{D^3\}$ operation.
- Smart PCA is $\mathcal{O}\{TNKD\}$, where K is the number of principal components you want to find, T is the number of training epochs, and N is the number of training tokens.
- If $K \ll D$ (e.g., you have 100,000 dimensions but you only want 100 principal components), smart PCA can be faster than regular PCA.



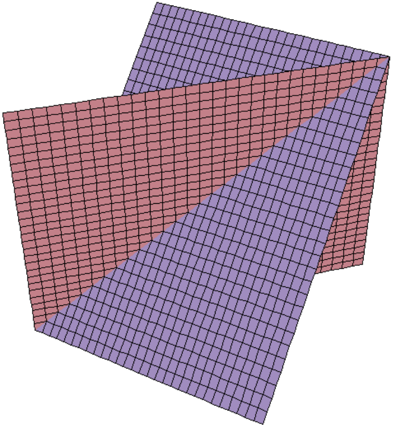
`https://commons.wikimedia.org/wiki/File:Autoencoder_schema.png`

Types of autoencoders

- A **linear autoencoder** finds the first K principal components. This architecture is sometimes called smart PCA, because it is faster than PCA if $K \ll D$.
- A **sparse autoencoder** can represent data that are drawn from a **sparse manifold**, which is the union of two or more hyperplanes.
- A **deep autoencoder** can represent data that are drawn from a **nonlinear manifold**, which is a curved lower-dimensional space embedded in a higher dimensional space.
- An **LSTM autoencoder** represents a sequence $[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T]$ using a shorter sequence $[\boldsymbol{h}_1, \ldots, \boldsymbol{h}_U]$ $(U < T)$.

Sparse Manifold

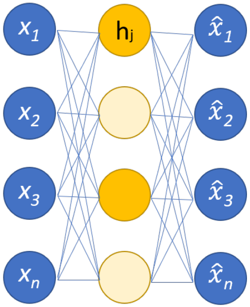
A sparse manifold is the union of two or more hyperplanes.
For example, suppose that we have a set of 3d vectors, all of which are drawn from one of the two planes shown at right. How can we represent that?



https://commons.wikimedia.org/wiki/File:PlaneIntersection.png

Sparse Autoencoder/Sparse PCA

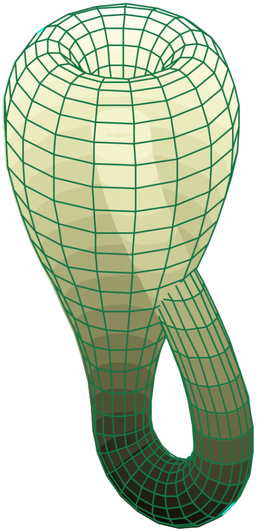
- A **sparse autoencoder** is an autoencoder in which $\text{len}(\boldsymbol{h}) \geq \text{len}(\boldsymbol{x})$, but only the $K < \text{len}(\boldsymbol{x})$ largest elements of $\boldsymbol{h}$ are allowed to be nonzero.
- For example, in the image at right, only the two largest elements of $\boldsymbol{h}$ are nonzero; the others are zeroed out.
- The result is that $\boldsymbol{x}' = \boldsymbol{x}$ if $\boldsymbol{x}$ is on a plane spanned by any two of the columns of $\boldsymbol{W}$.



`https://commons.wikimedia.org/wiki/File:Autoencoder_sparso.png`

Nonlinear Manifold

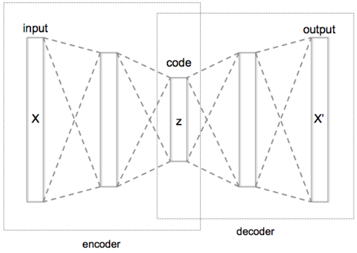
A nonlinear manifold is a curved lower-dimensional surface embedded in a higher-dimensional vector space. For example, the figure at right shows a 2d manifold embedded in a 3d vector space.



`https://commons.wikimedia.org/wiki/File:`

Deep Autoencoder

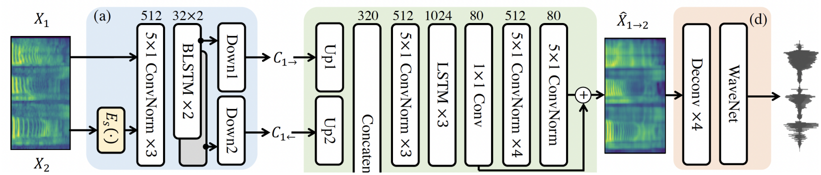
In a deep autoencoder, $\boldsymbol{z} = f(\boldsymbol{x})$ is computed by a network with at least two layers. Since a two-layer network can compute any function, therefore $f(\boldsymbol{x})$ can be any function.
The autoencoder loss can be zero ($\boldsymbol{x}' = \boldsymbol{x}$) whenever $\boldsymbol{x}$ is on a nonlinear manifold whose dimension is less than or equal to $\text{len}(\boldsymbol{z})$.



`https://commons.wikimedia.org/wiki/File:Autoencoder_structure.png`

LSTM Autoencoder

An LSTM autoencoder downsamples $[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T]$ to $[\boldsymbol{h}_1, \ldots, \boldsymbol{h}_U]$, then upsamples again to reconstruct $\boldsymbol{x}'$.



http://proceedings.mlr.press/v97/qian19c/qian19c.pdf

Outline

Voice Conversion

- Voice conversion transforms one person's speech so that it sounds like another person.
- Usually we transform one spectrogram to another, then use Griffin-Lim to reconstruct the waveform.

Fully Supervised Voice Conversion

- In fully-supervised voice conversion, we have examples of the source speaker and the target speaker saying the same sentences.
- A training example is $(\boldsymbol{X}_1, \boldsymbol{X}_2)$ where $\boldsymbol{X}_1 = [\boldsymbol{x}_{1,1}, \ldots, \boldsymbol{x}_{1,T}]$ is a source spectrogram and $\boldsymbol{X}_2 = [\boldsymbol{x}_{2,1}, \ldots, \boldsymbol{x}_{2,T}]$ is a target spectrogram (of another speaker saying the same thing). The first step is to approximately time-align them, e.g., by zero-padding the shorter one.
- Then we train a FCN, CNN, LSTM or other neural network that computes $\hat{\boldsymbol{x}}_{2,t} = f_t(\boldsymbol{X}_1)$, and train it to minimize

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{X}_2 - \boldsymbol{X}_{1\to 2}\|_F^2 = \frac{1}{2}\sum_{t=1}^{T}\|\boldsymbol{x}_{2,t} - f_t(\boldsymbol{X}_1)\|^2$$

Many-to-Many Voice Conversion

- A many-to-many voice converter is trained using a database with many speakers.
- The "content code" $\boldsymbol{C}_1 = E_C(\boldsymbol{X}_1)$ is a sequence of vectors specifying what the generated utterance should say.
- The "speaker code" $\boldsymbol{S}_2 = E_S(\boldsymbol{X}_2)$ specifies what the target speakers sounds like, e.g., it might be an x-vector or d-vector.
- The generated utterance $\hat{\boldsymbol{X}}_{1\rightarrow 2} = D(\boldsymbol{C}_1, \boldsymbol{S}_2)$ is a spectrogram with the same text content as $\boldsymbol{X}_1$, but sounds like the person who said $\boldsymbol{X}_2$.

Many-to-Many VC Examples

`https://auspicious3000.github.io/autovc-demo/`

How to train many-to-many voice conversion

- Fully supervised: We have a database (like VCTK) in which each speaker says the same things, so we can train using

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{X}_2 - \boldsymbol{X}_{1\rightarrow 2}\|_F^2$$

- StarGAN, AutoVC: We have a database of many speakers, but they are not all saying the same things, so we need to use some more clever training method.

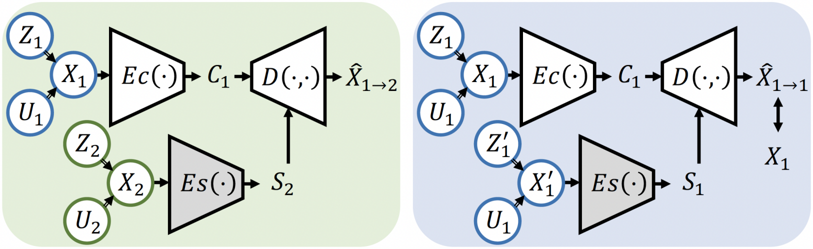
Outline

AutoVC: Zero-Shot Voice Conversion Using Autoencoder Loss

The key idea of AutoVC is that we train the network using self-reconstructions, $\boldsymbol{X}_{1\rightarrow 1} = D(\boldsymbol{C}_1, \boldsymbol{S}_1)$:

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{X}_1 - \boldsymbol{X}_{1\rightarrow 1}\|_F^2$$

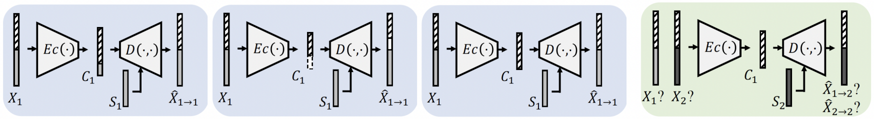
Then, during test time, we substitute in the speaker code for a different speaker.



AutoVC: How to avoid C_1 and S_1 swapping information?

- The problem that needs to be solved is: how do we make sure that $\boldsymbol{S}_1$ encodes **only** information about the speaker, and $\boldsymbol{C}_1$ encodes **only** information about the content?
- Suppose we pre-train the speaker encoder, $\boldsymbol{S}_1$, as a speaker verification system. The task of speaker verification works best if the utterance content has been normalized away, so training in this way will ensure that $\boldsymbol{S}_1$ contains no content information.
- Now we just need to remove any speaker information from the content code.

The “information bottleneck” idea



(a) Bottleneck too wide (b) Bottleneck too narrow (c) Bottleneck just right (d) Conversion

Figure 2. An intuitive explanation of how AUTOVC works. The target speaker is the same as the source speaker during training ((a)-(c)), and different during the actual conversion ((d)). Each speech segment contains two types of information: the speaker information (solid) and content information (striped). (a) When the bottleneck is too wide, the content embedding will contain some source speaker information. (b) When the bottleneck is too narrow, the content information is lost, which leads to imperfect reconstruction. (c) When the bottleneck is just right, perfect reconstruction is achievable, *and* the content embedding contains no source speaker information. (d) During the actual conversion, the output should contain no information about the source speaker, so the conversion quality should be as high as if it were doing self-reconstruction.

The "information bottleneck" idea

- The input spectrogram is $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T]$, where each spectral vector has a dimension of D.
- The content code is $\boldsymbol{C} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_U]$, where each content codevector has a dimension of K.
- The bottleneck factor, B, is the ratio of size($\boldsymbol{C}$)/size($\boldsymbol{X}$), i.e.,

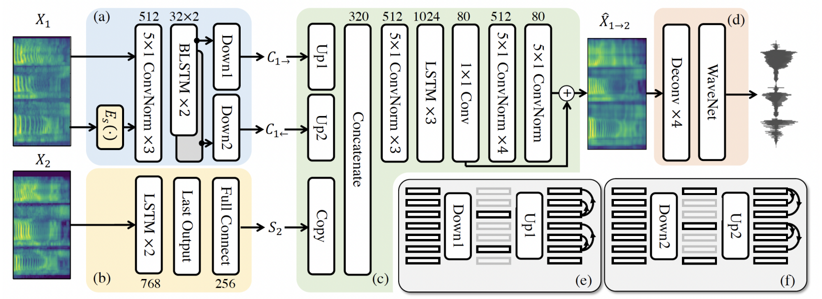
$$B = \left(\frac{K}{D}\right) \times \left(\frac{U}{T}\right)$$

The “information bottleneck” idea

$$B = \left(\frac{K}{D}\right) \times \left(\frac{U}{T}\right)$$

- When B is too large, speaker information leaks into the content code, so a converted speech file sounds like the source speaker. We can measure this using a speaker verification system: ask it, is this converted file the same as the source speaker? If so, reduce B, and train again.
- When B is too small, converted speech is unintelligible. We can test this by measuring $\mathcal{L} = \frac{1}{2}\|\boldsymbol{X}_1 - \boldsymbol{X}_{1\rightarrow 1}\|^2$. If $\mathcal{L}$ is too large, increase B, and train again.
- When B is just right, converted speech is intelligible, but sounds nothing like the source speaker.

The AutoVC Architecture



Zero-Shot Voice Conversion

Since AutoVC represents the target speaker using a d-vector, it can be used to convert speech into the voice of somebody it never heard during training ("zero-shot voice conversion"). Here are some examples:

`https://auspicious3000.github.io/autovc-demo/`

Outline

Conclusions: Autoencoders

- An autoencoder converts $\boldsymbol{x}$ into $\boldsymbol{h}$ such that $\text{len}(\boldsymbol{h}) < \text{len}(\boldsymbol{x})$, but $\boldsymbol{h}$ contains most of the information in $\boldsymbol{x}$, in the sense that it minimizes $\|\boldsymbol{x}' - \boldsymbol{x}\|^2$.
- A linear autoencoder finds PCA. The linear autoencoder is called "smart PCA" because, if $\text{len}(\boldsymbol{h}) \ll \text{len}(\boldsymbol{x})$, training the neural net may be less computationally expensive than finding the eigenvectors of the covariance.
- A sparse autoencoder permits at most K elements of $\boldsymbol{h}$ to be nonzero, and therefore achieves zero error if $\boldsymbol{x}$ is drawn from a K-dimensional sparse manifold.
- A deep autoencoder finds $\boldsymbol{z} = f(\boldsymbol{x})$, a nonlinear transformation of $\boldsymbol{x}$, and is able to represent $\boldsymbol{x}$ with nonzero error if $\boldsymbol{x}$ is drawn from a $\text{len}(\boldsymbol{z})$-dimensional nonlinear manifold.
- An LSTM autoencoder can convert between sequences of different lengths.

Conclusions: Voice Conversion

- Supervised voice conversion assumes that you have paired utterances, in which source and target speaker say the same thing.
  - One-to-one: one source speaker, one target speaker.
  - Many-to-many: target speaker is specified by a speaker-ID vector, e.g., a d-vector. System is trained to generate a voice given its d-vector.
- Zero-shot voice conversion assumes that the target speaker was not part of the training dataset.
  - AutoVC is trained using self-conversion, i.e., $\boldsymbol{X}_{1\rightarrow 1} = D(\boldsymbol{C}_1, \boldsymbol{S}_1)$.
  - The speaker code is a speaker-ID system, e.g., d-vector.
  - If speaker verification can tell who the source speaker was, then reduce the bottleneck dimension and try again.
  - If reconstruction error rate is too high, then increase the bottleneck dimension and try again.