Lecture 21

Autoencoders

\[
\text{minimize loss function: } \\
L(\hat{x}, \hat{g}(f(x)))
\]

\[
\text{Learn } \hat{g}(f(x)) = \hat{x} \text{ copies input to output}
\]

We're interested in the code \( h \):

dimensionality reduction, feature learning

We need to avoid overfitting (i.e., just copying input to output):

- use undercomplete autoencoder (code dim' < input dim')
- use overcomplete autoencoder (Dim < code dim')

+ regularization for sparsity of the code, robustness to noise, robustness to translations/rotations, etc.

**Sparse autoencoders:**

\[
L(\hat{x}, \hat{g}(f(x))) + \lambda \| h \|
\]

Could be used e.g. for feature learning which then provide input to classification. For example, one could use ReLU activation units.
in the code layer \( \oplus \) \( R(h) = 1 \使用者 \text{ penalty.} \)

This should produce \( 0 \)'s in the \( h \) code.

\textbf{Denoising autoencoders:} minimize

\[ L(\tilde{x}, \tilde{y}(\tilde{f}(\tilde{x}))) \text{, where} \]
\[ \tilde{x} = x + \tilde{\xi} \text{ is perturbed input noise} \]

This forces autoencoders to undo the noise corruption.

\textbf{Contractive autoencoders:} minimize

\[ L(\tilde{x}, \tilde{y}(\tilde{f}(\tilde{x}))) + \lambda \sum_i \| \frac{\partial}{\partial x_i} h_i \| ^2 \text{ where} \]
\[ h_i = f_i(\tilde{x}) \]

Autoencoders → shallow (as discussed thus far)

\( \text{Deep} \) (multiple layers to represent \( f \) & \( \tilde{f} \)):
Ex.: linear autoencoder

\[ \tilde{h} = W_f \tilde{x}, \quad \tilde{x} = W_g \tilde{h} \]

\[ \tilde{x} \quad \tilde{h} \quad \tilde{\tilde{x}} = \tilde{x} \]

\[ D_1 \quad D_2 \leq D_1 \quad D_1 \]

\[
\min_{w_f, w_g} \left\{ \frac{1}{2} \sum_{n} \left| w_g w_f \tilde{x}_n - \bar{x}_n \right|^2 \right\}
\]

\[ L = \text{loss function} \]

Equivalent to PCA (principal component analysis)

\[ \text{non-linear autoencoder} \]

\[
\min_{w_f, w_g} \left\{ \frac{1}{2} \sum_{n} \left| \tilde{y}(f(x_n; w_f); w_g) - \bar{x}_n \right|^2 \right\}
\]

\[ \text{NN weights + biases} \]

[Projection onto lower-D non-linear manifold]
Probabilistic autoencoders

\[ f: \ \text{Pr}(h|x; W_f) \]
\[ g: \ \text{Pr}(\tilde{x}|h; W_g) \]

use sigmoid, softmax or linear units at the code & output layers to produce Gaussian \( \mu, \sigma \), discard "-" if negative

\( P(\tilde{x}|h; W_g) \)

\( \tilde{x}, \tilde{x}', h \) are random vars

Can use these as generative models to produce novel outputs:

1. Sample \( h \) from \( P(h) \)
2. Sample \( \tilde{x} \) from decoder: \( P(\tilde{x}|h; W_g) \)
[Denoising autoencoders: ]
(detailed discussion)

1. Sample \( \tilde{x} \) from training data
2. Sample noisy/corrupted \( \tilde{x}' \): \( \tilde{x}' = x + \tilde{e} \Rightarrow P(\tilde{x}' | x) \)

3. Use \((\tilde{x}, \tilde{x}')\) to construct
   \[ P_{\text{decoder}} (\tilde{x} | h = f(\tilde{x}')) = P_{\text{reconstruct}} (\tilde{x} | \tilde{x}') \]
   \( \triangleright \) minimize
   \[ -\log P_{\text{decoder}} (\tilde{x} | h) \]

The error function is given by
\[ E = \sum_{x} \int d\tilde{x}' P(x) P(\tilde{x}' | x) \log P_{\text{decoder}} (\tilde{x} | h = f(x)) \]

"summed" over many datapoints \( x \& \tilde{x}' \), for each \( x \), many corrupted/noisy \( \tilde{x}' \)

original data on low-D manifold

\[ \tilde{x}' \]

contours of \( P(\tilde{x}' | x) \)
Now, consider quadratic error:

$$E = \int dx \int dx' \ p(x) \ P(x' \mid x) \ |g(f(x')) - \bar{x}|^2$$

learns to be as close to \( \bar{x} \) as possible, by averaging over many \( x' \) realizations

points towards the nearest point on the manifold

\( \bar{x} - x' = \)

represents a gradient flow towards the "true" \( \bar{x} \)-manifold