Neural Networks

Lecture 11: Autoencoders & GANs
<table>
<thead>
<tr>
<th>Lesson</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Probability and Information Theory</td>
</tr>
<tr>
<td>3</td>
<td>Basics of Machine Learning</td>
</tr>
<tr>
<td>4</td>
<td>Deep Feed-forward networks</td>
</tr>
<tr>
<td>5</td>
<td>Back-propagation</td>
</tr>
<tr>
<td>6</td>
<td>Optimization and regularization</td>
</tr>
<tr>
<td>7</td>
<td>Convolutional Networks</td>
</tr>
<tr>
<td>8</td>
<td>Sequence Modeling: recurrent networks</td>
</tr>
<tr>
<td>9</td>
<td>Applications</td>
</tr>
<tr>
<td>10</td>
<td>Software and Practical methods</td>
</tr>
<tr>
<td>11</td>
<td>Representational Learning</td>
</tr>
<tr>
<td>12</td>
<td>Deep Generative Models</td>
</tr>
<tr>
<td>13</td>
<td>Deep Reinforcement Learning</td>
</tr>
<tr>
<td>14</td>
<td>Deep Learning and the Brain</td>
</tr>
<tr>
<td>15</td>
<td>Conclusions and Future Perspective</td>
</tr>
<tr>
<td>16</td>
<td>Project presentations</td>
</tr>
</tbody>
</table>

**Foundations**

**Basics**

**Advanced**

**Selected topics**
<table>
<thead>
<tr>
<th>Lesson</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Probability and Information Theory</td>
</tr>
<tr>
<td>3</td>
<td>Basics of Machine Learning</td>
</tr>
<tr>
<td>4</td>
<td>Deep Feed-forward networks</td>
</tr>
<tr>
<td>5</td>
<td>Back-propagation</td>
</tr>
<tr>
<td>6</td>
<td>Optimization and regularization</td>
</tr>
<tr>
<td>7</td>
<td>Convolutional Networks</td>
</tr>
<tr>
<td>8</td>
<td>Sequence Modeling: recurrent networks</td>
</tr>
<tr>
<td>9</td>
<td>Applications</td>
</tr>
<tr>
<td>10</td>
<td>Software and Practical methods</td>
</tr>
<tr>
<td>11</td>
<td>Autoencoders &amp; GANs</td>
</tr>
<tr>
<td>12</td>
<td>Project Development</td>
</tr>
<tr>
<td>13</td>
<td>Deep Reinforcement Learning</td>
</tr>
<tr>
<td>14</td>
<td>Deep Learning and the Brain</td>
</tr>
<tr>
<td>15</td>
<td>Conclusions and Future Perspective</td>
</tr>
<tr>
<td>16</td>
<td>Project presentations</td>
</tr>
</tbody>
</table>
Learning objectives

• Autoencoders
• Transfer learning and domain adaptation
• Generative adversarial networks
Types of ML depending of experience

- **Supervised**: each sample is associated to a label/target

\[
P(y|x)
\]

\[
\text{classifier } \quad f: X \rightarrow Y
\]

\[
\{\text{“cat”, “dog”}\}
\]

\[
P(y|x)
\]
Supervised learning

**Data:** \((x, y)\)
x is data, \(y\) is label

**Goal:** Learn a function to map \(x \rightarrow y\)

**Examples:** classification, regression, object detection, segmentation, image captioning, etc.
Types of ML depending of experience

- Supervised
- Unsupervised: each sample contains features but no label

\[ P(x) \]
Unsupervised learning

**Data:** $x$
just data, no labels

**Goal:** Learn some underlying hidden structure of the data

**Examples:** clustering, dimensionality reduction, feature learning, density estimation, etc.

K-means clustering
Unsupervised learning

Data: \( x \)
just data, no labels

Goal: Learn some underlying hidden structure of the data

Examples: clustering, dimensionality reduction, feature learning, density estimation, etc.

Principal component analysis
(Dimensionality reduction)
Unsupervised learning

**Data:** \( x \)
just data, no labels

**Goal:** Learn some underlying hidden structure of the data

**Examples:** clustering, dimensionality reduction, feature learning, density estimation, etc.
Unsupervised learning

**Data:** $x$
just data, no labels

**Goal:** Learn some underlying hidden structure of the data

**Examples:** clustering, dimensionality reduction, feature learning, density estimation, etc.

2-d density estimation
Types of ML depending of experience

- Supervised
- Unsupervised
- Reinforcement: experience via interacting with environment
### Supervised vs Unsupervised learning

<table>
<thead>
<tr>
<th>Supervised learning</th>
<th>Unsupervised learning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data:</strong> $(x, y)$</td>
<td><strong>Data:</strong> $x$</td>
</tr>
<tr>
<td>$x$ is data, $y$ is label</td>
<td>just data, no labels!</td>
</tr>
<tr>
<td><strong>Goal:</strong> Learn a function</td>
<td><strong>Goal:</strong> Learn some underlying hidden structure of the data</td>
</tr>
<tr>
<td>to map $x \rightarrow y$</td>
<td></td>
</tr>
<tr>
<td><strong>Examples:</strong> classification, regression, object detection, segmentation, image captioning, etc.</td>
<td><strong>Examples:</strong> clustering, dimensionality reduction, feature learning, density estimation, etc.</td>
</tr>
</tbody>
</table>
Supervised vs Unsupervised learning

**Supervised learning**

**Data:** \((x, y)\)
- \(x\) is data, \(y\) is label

**Goal:** Learn a function to map \(x \rightarrow y\)

**Examples:** classification, regression, object detection, segmentation, image captioning, etc.

**Unsupervised learning**

**Data:** \(x\)
- just data, no labels!

**Goal:** Learn some underlying hidden structure of the data

**Examples:** clustering, dimensionality reduction, feature learning, density estimation, etc.

Training data is cheap
Supervised vs Unsupervised learning

**Supervised learning**

**Data:** $(x, y)$
$x$ is data, $y$ is label

**Goal:** Learn a function to map $x \rightarrow y$

**Examples:** classification, regression, object detection, segmentation, image captioning, etc.

**Unsupervised learning**

**Data:** $x$
just data, no labels!

**Goal:** Learn some underlying hidden structure of the data

**Examples:** clustering, dimensionality reduction, feature learning, density estimation, etc.

Training data is cheap

Holy grail: solve unsupervised learning $\rightarrow$ understand structure generating process
**Supervised learning**

Data: $(x, y)$
$x$ is data, $y$ is label

Goal: Learn a function to map $x \rightarrow y$

Examples: classification, regression, object detection, segmentation, image captioning, etc.

**Unsupervised learning**

Data: $x$
just data, no labels!

Goal: Learn some underlying hidden structure of the data

Examples: clustering, dimensionality reduction, feature learning, density estimation, etc.

Training data is cheap

Holy grail: solve unsupervised learning -> understand structure generating process
Supervised vs Unsupervised learning

**Supervised learning**

**Data:** \((x, y)\)

- \(x\) is data, \(y\) is label

**Goal:** Learn a function to map \(x \rightarrow y\)

**Examples:** classification, regression, object detection, segmentation, image captioning, etc.

**Unsupervised learning**

**Data:** \(x\)

- just data, no labels!

**Goal:** Learn some underlying hidden structure of the data

**Examples:** clustering, dimensionality reduction, feature learning, density estimation, etc.

Training data is cheap

Holy grail: solve unsupervised learning -> understand structure generating process
Supervised vs Unsupervised learning

**Supervised learning**

**Data:** \((x, y)\)
- \(x\) is data, \(y\) is label

**Goal:** Learn a function to map \(x \rightarrow y\)

**Examples:** classification, regression, object detection, segmentation, image captioning, etc.

**Unsupervised learning**

**Data:** \(x\)
- just data, no labels!

**Goal:** Learn some underlying hidden structure of the data

**Examples:** clustering, dimensionality reduction, feature learning, density estimation, etc.

**Training data is cheap**

**Holy grail:** solve unsupervised learning -> understand structure generating process

**Autoencoders**

**GANs**
Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabelled training data

$z$ usually smaller than $x$

(dimENSIONALITY reduction)
Autoencoder:

Unsupervised approach for learning a lower-dimensional feature representation from unlabelled training data.

- **Originally**: Linear + nonlinearity (sigmoid)
- **Later**: Deep, fully connected
- **Later**: ReLU CNN
Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabelled training data

$z$ usually smaller than $x$ (dimensionality reduction)

Why smaller?

- **Originally:** Linear + nonlinearity (sigmoid)
- **Later:** Deep, fully connected
- **Later:** ReLU CNN
Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabelled training data

\( z \) usually smaller than \( x \) (dimensionality reduction)

Why smaller?
We want features to capture meaningful factors of variation in data

**Originally:** Linear + nonlinearity (sigmoid)

**Later:** Deep, fully connected

**Later:** ReLU CNN

"..."
Autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself
Autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding" - encoding itself

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully connected
Later: ReLU CNN (upconv)
Autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself
Autoencoders

Train such that features can be used to reconstruct original data
Autoencoders

Train such that features can be used to reconstruct original data. Doesn't use labels!
Autoencoders

After training, throw away the decoder!
Autoencoders
Autoencoders

Encoder can be used to initialize a **supervised** model.

![Diagram of Autoencoders](image)

- **Input data** \( \mathbf{x} \)
- **Features** \( \mathbf{z} \)
- **Encoder**
- **Classifier**
- **Predicted Label** \( \hat{y} \)
- **True Label** \( y \)

**Loss function** (Softmax, etc)

Fine-tune encoder jointly with classifier.
Autoencoders

Encoder can be used to initialize a **supervised** model.

- Loss function (Softmax, etc)
- Predicted Label \( \hat{y} \)
- Classifier
- Features \( z \)
- Encoder
- Input data \( x \)

Fine-tune encoder jointly with classifier

Train for final task (sometimes with small data)

- bird
- plane
- dog
- deer
- truck
Transfer learning (sharing representations)

One representation used for many tasks or input formats
Transfer learning (sharing representations)

One representation used for many tasks or input formats

Task 1: women vs men

Task 2: young vs old

Input: images
Transfer learning (sharing representations)

One representation used for many tasks or input formats

Task: sentiment analysis

Input 1: book reviews
Input 2: music reviews
Autoencoders for generating data

Autoencoders can reconstruct data, and can learn features to initialize a supervised model.

Features capture factors of variation in training data. Can we generate new images from an auto encoder?
Autoencoders for generating data

Autoencoders can reconstruct data, and can learn features to initialize a supervised model.

Features capture factors of variation in training data. Can we generate new images from an auto encoder?

Intuition: $x$ is an image, $z$ is latent factors used to generate $x$: attributes, orientation, etc.
Autoencoders for generating data

Assume training data is generated from underlying unobserved (latent) representation $z$

Obtain $p(x|z)$

Sample from some prior $p(z)$

Autoencoders can reconstruct data, and can learn features to initialize a supervised model.

Features capture factors of variation in training data. Can we generate new images from an auto encoder?

**Intuition:** $x$ is an image, $z$ is latent factors used to generate $x$: attributes, orientation, etc.
Autoencoders for generating data

Assume training data is generated from underlying unobserved (latent) representation $z$

Different dimensions of $z$ encode interpretable factors of variation
Autoencoders for generating data

Assume training data is generated from underlying unobserved (latent) representation $z$
Mean Squared Error Can Ignore Small but Task-Relevant Features (Autoencoders)

The ping pong ball vanishes because it is not large enough to significantly affect the mean squared error.
Generative adversarial networks

**Problem:** want to sample from complex, high-dimensional training distribution. No direct way to do this!

**Solution:** sample from a simple distribution, e.g. random noise. Learn transformation to target distribution.

What can we use to represent this complex transformation?
Generative adversarial networks

**Problem:** want to sample from complex, high-dimensional training distribution. No direct way to do this!

**Solution:** sample from a simple distribution, e.g. random noise. Learn transformation to target distribution.

What can we use to represent this complex transformation?

A neural network!

Output: sample from training distribution

Input: Random noise
Training GANs: Two-player game

**Generator network:** try to fool the discriminator by generating real-looking pictures

**Discriminator network:** try to distinguish between real and fake images
Training GANs: Two-player game

**Generator network:** try to fool the discriminator by generating real-looking pictures

**Discriminator network:** try to distinguish between real and fake images
Training GANs: Two-player game

**Generator network:** try to fool the discriminator by generating real-looking pictures

**Discriminator network:** try to distinguish between real and fake images

Train jointly in minimax game

Minimax objective function:

\[
\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]
Training GANs: Two-player game

**Generator network:** try to fool the discriminator by generating real-looking pictures

**Discriminator network:** try to distinguish between real and fake images

Train jointly in minimax game

Minimax objective function:

\[
\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]
Training GANs: Two-player game

**Generator network:** try to fool the discriminator by generating real-looking pictures

**Discriminator network:** try to distinguish between real and fake images

Train jointly in minimax game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

- **Discriminator** wants to **maximize objective** such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake)

- **Generator** wants to **minimize objective** such that $D(G(z))$ is close to 1 (discriminator is fooled into thinking generated $G(z)$ is real)
Training GANs: Two-player game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. Gradient ascent on discriminator

$$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. Gradient descent on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z)))$$
Training GANs: Two-player game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. Gradient ascent on discriminator

$$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead**: Gradient ascent on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log (D_{\theta_d}(G_{\theta_g}(z)))$$
Training GANs: Two-player game

Putting it together: GAN training algorithm

for number of training iterations do
  for k steps do
    • Sample minibatch of m noise samples \( \{z^{(1)}, \ldots, z^{(m)}\} \) from noise prior \( p_g(z) \).
    • Sample minibatch of m examples \( \{x^{(1)}, \ldots, x^{(m)}\} \) from data generating distribution \( p_{data}(x) \).
    • Update the discriminator by ascending its stochastic gradient:
      \[
      \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right] 
      \]
  end for
  • Sample minibatch of m noise samples \( \{z^{(1)}, \ldots, z^{(m)}\} \) from noise prior \( p_g(z) \).
  • Update the generator by ascending its stochastic gradient (improved objective):
    \[
    \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)}))) 
    \]
end for
**Training GANs:** Two-player game

- **Generator network:** try to fool the discriminator by generating real-looking pictures
- **Discriminator network:** try to distinguish between real and fake images

Diagram:
- **Fake Images** (from generator) to **Generator Network**
- **Random noise** $z$ to **Generator Network**
- **Real Images** (from training set) to **Discriminator Network**
- **Discriminator Network** outputs **Real or Fake**
- **After training, use generator network to generate new images**
Adversarial Losses Preserve Any Features with Highly Structured Patterns

Mean squared error loses the ear because it causes a small change in few pixels. Adversarial loss preserves the ear because it is easy to notice its absence.
GANs

Generated samples

Nearest neighbours from training set
GANs

Generated samples
GANs learn vector spaces that support semantic arithmetic

Figure 15.9: A generative model has learned a distributed representation that disentangles the concept of gender from the concept of wearing glasses. If we begin with the representation of the concept of a man with glasses, then subtract the vector representing the concept of a man without glasses, and finally add the vector representing the concept of a woman without glasses, we obtain the vector representing the concept of a woman with glasses. The generative model correctly decodes all these representation vectors to images that may be recognized as belonging to the correct class. Images reproduced with permission from Radford et al. (2015).

There is no need to have labels for the hidden unit classifiers: gradient descent on an objective function of interest naturally learns semantically interesting features, as long as the task requires such features. We can learn about the distinction between male and female, or about the presence or absence of glasses, without having to characterize all the configurations of the other features by examples covering all these combinations of values. This form of statistical separability is what allows one to generalize to new configurations of a person's features that have never been seen during training.

15.5 Exponential Gains from Depth

We have seen in section 6.4.1 that multilayer perceptrons are universal approximators, and that some functions can be represented by exponentially smaller deep networks compared to shallow networks. This decrease in model size leads to improved statistical efficiency. In this section, we describe how similar results apply more generally to other kinds of models with distributed hidden representations.
GANs learn vector spaces that support semantic arithmetic

Figure 15.9: A generative model has learned a distributed representation that disentangles the concept of gender from the concept of wearing glasses. If we begin with the representation of the concept of a man with glasses, then subtract the vector representing the concept of a man without glasses, and finally add the vector representing the concept of a woman without glasses, we obtain the vector representing the concept of a woman with glasses. The generative model correctly decodes all these representation vectors to images that may be recognized as belonging to the correct class. Images reproduced with permission from Radford et al. (2015).
Many GAN applications

Source-> Target domain transfer

https://affinelaye...
edges2shoes

TOOL
- line
- eraser

INPUT

OUTPUT

undo  clear  random

pix2pix

process

save
edges2handbags

TOOL
- line
- eraser

INPUT
- undo
- clear
- random

OUTPUT
- pix2pix process
- save
edges2cats

TOOL
- line
- eraser

INPUT

OUTPUT

undo  clear  random

pix2pix
process

save
<table>
<thead>
<tr>
<th>Lesson</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Probability and Information Theory</td>
</tr>
<tr>
<td>3</td>
<td>Basics of Machine Learning</td>
</tr>
<tr>
<td>4</td>
<td>Deep Feed-forward networks</td>
</tr>
<tr>
<td>5</td>
<td>Back-propagation</td>
</tr>
<tr>
<td>6</td>
<td>Optimization and regularization</td>
</tr>
<tr>
<td>7</td>
<td>Convolutional Networks</td>
</tr>
<tr>
<td>8</td>
<td>Sequence Modeling: recurrent networks</td>
</tr>
<tr>
<td>9</td>
<td>Applications</td>
</tr>
<tr>
<td>10</td>
<td>Software and Practical methods</td>
</tr>
<tr>
<td>11</td>
<td>Autoencoders &amp; GANs</td>
</tr>
<tr>
<td>12</td>
<td>Project Development</td>
</tr>
<tr>
<td>13</td>
<td>Deep Reinforcement Learning</td>
</tr>
<tr>
<td>14</td>
<td>Deep Learning and the Brain</td>
</tr>
<tr>
<td>15</td>
<td>Conclusions and Future Perspective</td>
</tr>
<tr>
<td>16</td>
<td>Project presentations</td>
</tr>
</tbody>
</table>
CHAPTER 14. AUTOENCODERS

Recirculation is regarded as more biologically plausible than back-propagation, but is rarely used for machine learning applications.

Figure 14.1: The general structure of an autoencoder, mapping an input $x$ to an output $r$ through an internal representation or code $h$. The autoencoder has two components: the encoder $f$ (mapping $x$ to $h$) and the decoder $g$ (mapping $h$ to $r$).

14.1 Undercomplete Autoencoders

Copying the input to the output may sound useless, but we are typically not interested in the output of the decoder. Instead, 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 $x$. An autoencoder whose code dimension is less than the input dimension is called undercomplete. Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data.

The learning process is described simply as minimizing a loss function $L(x, g(f(x)))$ (14.1) where $L$ is a loss function penalizing $g(f(x))$ for being dissimilar from $x$, such as the mean squared error.

When the decoder is linear and $L$ is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA. In this case, an autoencoder trained to perform the copying task has learned the principal subspace of the training data as a side-effect.

Autoencoders with nonlinear encoder functions $f$ and nonlinear decoder functions $g$ can thus learn a more powerful nonlinear generalization of PCA. Unfortu...
Typically, the output variables are treated as being conditionally independent given $h$ so that this probability distribution is inexpensive to evaluate, but some techniques such as mixture density outputs allow tractable modeling of outputs with correlations.

Figure 14.2: The structure of a stochastic autoencoder, in which both the encoder and the decoder are not simple functions but instead involve some noise injection, meaning that their output can be seen as sampled from a distribution, $p_{\text{encoder}}(h | x)$ for the encoder and $p_{\text{decoder}}(x | h)$ for the decoder.

To make a more radical departure from the feedforward networks we have seen previously, we can also generalize the notion of an encoding function $f(x)$ to an encoding distribution $p_{\text{encoder}}(h | x)$, as illustrated in figure 14.2.

Any latent variable model $p_{\text{model}}(h, x)$ defines a stochastic encoder $p_{\text{encoder}}(h | x) = p_{\text{model}}(h | x)$ (14.12) and a stochastic decoder $p_{\text{decoder}}(x | h) = p_{\text{model}}(x | h)$. (14.13)

In general, the encoder and decoder distributions are not necessarily conditional distributions compatible with a unique joint distribution $p_{\text{model}}(x, h)$.

14.5 Denoising Autoencoders

The denoising autoencoder (DAE) is an autoencoder that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output.

The DAE training procedure is illustrated in figure 14.3. We introduce a corruption process $C(\tilde{x} | x)$ which represents a conditional distribution over $\tilde{x}$.

Structure of an autoencoder

$$p_{\text{encoder}}(h | x) \quad p_{\text{decoder}}(x | h)$$

(Goodfellow 2016)
Avoiding trivial identity

• **Undercomplete autoencoders**
  - $h$ has lower dimension than $x$
  - $f$ or $g$ has low capacity (e.g., linear $g$)
  - Must discard some information in $h$

• **Overcomplete autoencoders**
  - $h$ has higher dimension than $x$
  - Must be regularized
Regularized autoencoders

- Sparse autoencoders
- **Denoising autoencoders**
- Autoencoders with dropout on the hidden layer
- Contractive autoencoders
Denoising autoencoder

\[ C(\tilde{x} | x) \]

\[ L = - \log p_{\text{decoder}}(x | h = f(\tilde{x})) \]
Denoising autoencoder learns a manifold