Introduction

Autoencoding is training a network to replicate its input to its output

Applications:
- Unlabeled pre-training for semi-supervised learning
- Learning embeddings to support information retrieval
- Generation of new instances similar to those in the training set
- Data compression

Outline

Basic idea
- Stacking
- Types of autoencoders
  - Denoising
  - Sparse
  - Contractive
  - Variational
  - Generative adversarial networks

Basic Idea

- Autoencoder is a network trained to learn the identity function: output = input
- Subnetwork called encoder \( f(\cdot) \) maps input to an embedded representation
- Subnetwork called decoder \( g(\cdot) \) maps back to input space
- Can be thought of as lossy compression of input
- Need to identify the important attributes of inputs to reproduce faithfully

Autoencoders based on size of hidden layer
- Undercomplete autoencoders have hidden layer size smaller than input layer size
  - Dimension of embedded space lower than that of input space
  - Cannot simply memorize training instances
- Overcomplete autoencoders have much larger hidden layer sizes
  - Regularize to avoid overfitting, e.g., enforce a sparsity constraint
Basic Idea

Example: Principal Component Analysis

A 3-2-3 autoencoder with linear units and square loss performs principal component analysis: Find linear transformation of data to maximize variance.

Stacked Autoencoders

Incremental Training

Phase 1 Train the first autoencoder
Phase 2 Train the second autoencoder
Phase 3 Stack the autoencoders

- Can simplify training by starting with single hidden layer $H_1$
- Then, train a second AE to mimic the output of $H_1$
- Insert this into first network
- Can build by using $H_1$’s output as training set for Phase 2

Visualization

Input MNIST Digit

Network Output

Weights (features selected) for five nodes from $H_1$:

Stacked Autoencoders

Semi-Supervised Learning

- Can pre-train network with unlabeled data
  - learn useful features and then train “logic” of dense layer with labeled data
**Transfer Learning from Trained Classifier**

- Can also transfer from a classifier trained on different task, e.g., transfer a GoogleNet architecture to ultrasound classification.

- Often choose existing one from a model zoo.

---

**Denoising Autoencoders**

**Example**

- Can train an autoencoder to learn to denoise input by giving input corrupted instance $\tilde{x}$ and targeting uncorrupted instance $x$.

- Example noise models:
  - Gaussian noise: $\tilde{x} = x + z$, where $z \sim N(0, \sigma^2 I)$
  - Masking noise: zero out some fraction $\nu$ of components of $x$
  - Salt-and-pepper noise: choose some fraction $\nu$ of components of $x$ and set each to its min or max value (equally likely).

---

**Denoising Autoencoders**

- How does it work?
  - Even though, e.g., MNIST data are in a 784-dimensional space, they lie on a low-dimensional manifold that captures their most important features.
  - Corruption process moves instance $x$ off of manifold.
  - Encoder $f_h$ and decoder $g_{\phi^*}$ are trained to project $\tilde{x}$ back onto manifold.

---

**Sparse Autoencoders**

- An overcomplete architecture.
  - Regularize outputs of hidden layer to enforce sparsity:
    $$ \tilde{J}(x) = J(x, g(f(x))) + \alpha \Omega(h) $$
    where $J$ is loss function, $f$ is encoder, $g$ is decoder, $h = f(x)$, and $\Omega$ penalizes non-sparsity of $h$.
  - E.g., can use $\Omega(h) = \sum |h|$ and ReLU activation to force many zero outputs in hidden layer.
  - Can also measure average activation of $h_i$ across mini-batch and compare it to user-specified target sparsity $p$, e.g., $0.1$ via square error or Kullback-Leibler divergence:
    $$ p \log \frac{p}{q} + (1 - p) \log \frac{1 - p}{1 - q} $$
    where $q$ is average activation of $h_i$ over mini-batch.
Variational Autoencoders

Latent Variables

- Independence of \( z \) dimensions makes it easy to generate instances wrt complex distributions via decoder \( g \).
- Latent variables can be thought of as values of attributes describing inputs. For example, for MNIST, latent variables might represent "thickness", "slant", "loop closure".

Variational Autoencoders

Optimization

- Maximum likelihood (ML) approach for training generative models: find a model \( \theta \) with maximum probability of generating the training set \( \mathcal{X} \).
- Achieve this by minimizing the sum of:
  - End-to-end AE loss (e.g., square, cross-entropy)
  - Regularizer measuring distance (K-L divergence) from latent distribution \( q(z | x) \) and \( \mathcal{N}(0, I) \) (standard multivariate Gaussian).
  - \( \mathcal{N}(0, I) \) also considered the prior distribution over \( z \) (distribution when no \( x \) is known).

```
eps = 1e-10
latent_loss = 0.5 * tf.reduce_sum([tf.square(hidden3_sigma) + tf.square(hidden3_mean) - 1 - tf.log(eps + tf.square(hidden3_sigma))])
```

Variational Autoencoders

Reparameterization Trick

- Cannot backprop error signal through random samples.
- Reparameterization trick emulates \( z \sim \mathcal{N}(\mu, \sigma) \) with \( \epsilon \sim \mathcal{N}(0, 1) \), \( z = \epsilon \sigma + \mu \).
Variational Autoencoders
Example Generated Images: Random

- Draw $z \sim \mathcal{N}(0, I)$ and display $g(z)$

Variational Autoencoders
Example Generated Images: Manifold

- Uniformly sample points in $z$ space and decode

Variational Autoencoders
2D Cluster Analysis

- Cluster analysis by digit

Generative Adversarial Network

- GANs are also generative models, like VAEs
- Models a game between two players
  - Generator creates samples intended to come from training distribution
  - Discriminator attempts to discern the "real" (original training) samples from the "fake" (generated) ones
- Discriminator trains as a binary classifier, generator trains to fool the discriminator

Generative Adversarial Network
How the Game Works

- Let $D(x)$ be discriminator parameterized by $\theta^{(D)}$
  - Goal: Find $\theta^{(D)}$ minimizing $J(D) (\theta^{(D)}, \theta^{(G)})$
- Let $G(z)$ be generator parameterized by $\theta^{(G)}$
  - Goal: Find $\theta^{(G)}$ minimizing $J(G) (\theta^{(D)}, \theta^{(G)})$
- A Nash equilibrium of this game is $(\theta^{(D)}, \theta^{(G)})$ such that each $\theta^{(i)}, i \in \{D, G\}$ yields a local minimum of its corresponding $J$

Generative Adversarial Network
Training

- Each training step:
  - Draw a minibatch of $x$ values from dataset
  - Draw a minibatch of $z$ values from prior (e.g., $\mathcal{N}(0, I)$)
  - Simultaneously update $\theta^{(G)}$ to reduce $J^{(G)}$ and $\theta^{(D)}$ to reduce $J^{(D)}$, via, e.g., Adam
  - For $J^{(D)}$, common to use cross-entropy where label is 1 for real and 0 for fake
  - Since generator wants to trick discriminator, can use $J^{(G)} = -J^{(D)}$
  - Others exist that are generally better in practice, e.g., based on ML
Generative Adversarial Network
DCGAN: Radford et al. (2015)

- "Deep, convolution GAN"
- Generator uses transposed convolutions (e.g., `tf.layers.conv2d_transpose`) without pooling to upsample images for input to discriminator

DCGAN Generated Images: Bedrooms
Trained from LSUN dataset, sampled z space

DCGAN Generated Images: Adele Facial Expressions
Trained from frame grabs of interview, sampled z space

DCGAN Generated Images: Latent Space Arithmetic
Performed semantic arithmetic in z space!
(Non-center images have noise added in z space; center is noise-free)