Chapter 14 - Auto encoders

김정주(haje01@gmail.com)

- autoencoder는 input과 유사한 output을 출력하는 neural net이다.
- 이를 위한 $code$를 표현하는 히든레이어 h를 가짐

이 네트워크는 두 파트로 이루어짐

- encoding 함수 $\boldsymbol{h} = f(\boldsymbol{x})$
- decoding 함수 $\boldsymbol{r} = g(\boldsymbol{h})$

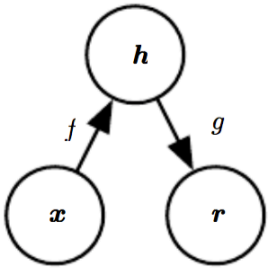


Figure 14.1: The general structure of an autoencoder, mapping an input $\boldsymbol{x}$ to an output (called reconstruction) $\boldsymbol{r}$ through an internal representation or code $\boldsymbol{h}$. The autoencoder has two components: the encoder f (mapping $\boldsymbol{x}$ to $\boldsymbol{h}$) and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

- 단순히 복사하기 위한 용도 즉, $g(f(\boldsymbol{x})) = \boldsymbol{x}$ 이라면 유용하지 않을 것
- autoencoder는 완벽하게 복사하는 것을 배우지 못하도록 설계되었다.
- autoencoder는 근사적으로, 또 학습 데이터와 유사한 것만을 copy하도록 제약을 받는다.
- 입력의 어떤 부분이 복사될지 우선순위를 조절하기에, 데이터에 관한 유용한 특성을 배움

- 최신 autoencoder는 encoder와 decoder를 결정(deterministic)함수에서 확률(stochastic) 매핑 즉, $p_{encoder}(\boldsymbol{h}|\boldsymbol{x})$ 와 $p_{decoder}(\boldsymbol{x}|\boldsymbol{h})$형태로 일반화 시켰다.

- 전통적으로 autoencoder는 차원 축소와 feature 학습에 사용되었지만
- 최근 잠재변수(latent variable) model과의 이론적 연관성때문에 생성(generative) model에서 중요한 역할을 하고있다.

- autoencoder는 feedforward 네트웍의 일종으로, 보통처럼 minibatch gradient descent 후 back-propagation으로 학습 하지만,
- 특이하게 입력값을 라벨로 하는 **recirculation**이라는 방법으로 학습될 수 있다.

14.1 Undercomplete Autoencoders

- input을 output으로 복사한다는 것은 의미가 없어 보인다.
- 우리는 output에 관심이 있는 것이 아니라, 학습 과정에서 h가 유용한 속성을 가지기를 기대한다.

- 유용한 특성을 가지도록 하는 방법 중 하나는 $\boldsymbol{h}$가 $\boldsymbol{x}$보다 작은 차원을 가지게 하는 것이다.
- autoencoder의 code 차원이 input 차원보다 작은 것을 **undercomplete**이라 한다.
- undercomplete하기에 input의 가장 현저한 특성을 배우게 된다.

학습 과정은 단순히 아래의 loss를 줄이는 과정이다.

$$L(\boldsymbol{x}, g(f(\boldsymbol{x})))$$

- decoder가 linear하고 L이 mean square error이면 undercomplte autoencoder는 PCA와 동일 subspace에 놓여진다.
- 이 경우 train data의 주부분공간(principal subspace_를 학습하게 된다.

- nonlinear encoder/decoder를 가지는 autoencoder는 더 강력한 nonlinear학습을 할수 있다.
- 그러나, encoder/decoder가 너무 많은 capacity를 가지면 copy 학습 과정에서 데이터의 중요한 특징을 배우지 못한다.

14.2 Regularized Autoencoders

- 비슷한 문제는 hidden code가 input dimension과 같거나 큰 경우(**overcomplete**)에도 일어난다.
- 이 경우에는 linear encoder/decoder라도 copy 학습 과정에서 데이터의 특징을 배우지 못한다.

- 이상적으로는 데이터의 복잡도에 맞춰 code 차원과 encoder/decoder capacity를 정하면 autoencoder학습은 항상 성공할 수 있다.
- Regularized autoencoder는 code 사이즈와 encoder/decoder capacity를 매번 선택하는 대신, 단순 copy가 아닌 다른 특징을 배울 수 있도록 해주는 loss 함수를 이용한다.
- Regularized autoencocer는 nonlinear이거나 overcomplete이어도, 심지어 model capacity가 커도 특징을 배울 수 있다.

- 여기에서 언급된 regularized autoencoder 방법외에도, latent variable과 inference procedure를 가진 generative model은 autoencoder의 특별한 형태로 볼 수 있는다.
  - descendants of the Helmholtz machine (Hinton, 1995b)
  - variational autoencoder (20.10.3)
  - generative stochastic networks (20.12)

- 이 모델들은 high-capacity, overcomplete encoding을 학습하면서도 regularization을 필요로 하지 않는다.
- 단순 copy가 아닌 확률을 최대화하는 encoder를 사용하기 때문이다.

14.2.1 Sparse Autoencoders

- Sparse autoencoder는 code 레이어 h에 대한 sparsity 페널티 $\Omega(\boldsymbol{h})$가 있는 autoencoder이다.

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h})$$

$g(\boldsymbol{h})$: decoder output, 일반적으로 $\boldsymbol{h} = f(\boldsymbol{x})$

- Sparse autoencoder는 classification 같은 좀 다른 테스크에 대한 특징 학습에 사용된다.
- Regularized되어 sparse 해진 autoencoder는, 단순히 identity function으로 동작하지 않고, train된 데이터 특유의 통계적 특성에 반응하게 된다.

- 우리는 Penalty $\Omega(\boldsymbol{h})$를 intput을 copy 하거나(자율학습 목표), sparse 피쳐에 의존하는 지도학습을 수행하는 feedforward network에 주어진 regularizer로 생각할 수 있다.
- 5.6.1에 설명한 것처럼 weight decay나 다른 regularization penalty를 수반한 학습은 베이즈 추론 (Bayesian inference)에서 모델 패러미터의 prior 확률 분포에 대한 regularized penalty가 추가된 MAP 추정으로 해석될 수 있다.
- 그 관점에서는 regularized maximum likelihood는 $p(\boldsymbol{\theta}|\boldsymbol{x})$를 최소화하는 것에 대응하고, 그것은 $\log p(\boldsymbol{x}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$를 최소화 하는 것과 같다.
  - $p(\boldsymbol{x}|\boldsymbol{\theta})$ 항은 보통의 data log-likelihood 항이고
  - $\log p(\boldsymbol{\theta})$항, 즉 패러미터에 대한 log-prior,은 $\boldsymbol{\theta}$의 특정 값에 대한 선호도를 구체화한다. (5.6 참고)
- Regularized autoencoder는 prior가 아닌 데이터에 의존하기에 그런 해석을 거부한다.

- sparsity penalty를 copy과정의 regularizer의 일종으로서의 생각하기 보다,
- 전체 sparse autoencoder 프레임워크를 latent variable을 가지는 generative model의 maximum likelihood 학습을 근사화하는 것으로 생각할 수 있다.

Suppose we have a model with visible variables $\boldsymbol{x}$ and latent variables $\boldsymbol{h}$, with an explicit joint distribution $p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h}) = p_{\text{model}}(\boldsymbol{h}) p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h})$. We refer to $p_{\text{model}}(\boldsymbol{h})$ as the model's prior distribution over the latent variables, representing the model's beliefs prior to seeing $\boldsymbol{x}$. This is different from the way we have previously used the word "prior," to refer to the distribution $p(\boldsymbol{\theta})$ encoding our beliefs about the model's parameters before we have seen the training data. The log-likelihood can be decomposed as

- 관측 변수 $\boldsymbol{x}$와 잠재 변수 $\boldsymbol{h}$에 대해 explicit joint distribution $p_{model}(\boldsymbol{x}, \boldsymbol{h}) = p_{model}(\boldsymbol{h}) p_{model}(\boldsymbol{x}|\boldsymbol{h})$을 가지는 모델이 있다고 하자
- $p_{model}(\boldsymbol{h})$은 잠재 변수에 대한 모델의 prior 분포인데,
- 그것은 $\boldsymbol{x}$를 보기 전(prior) model의 belief를 나타낸다.
- 이것은 우리가 학습 데이터를 보기 전 모델의 패러미터에 대한 belief를 담는 분포 $p(\boldsymbol{\theta})$를 나타내기 위한 단어 prior 와는 의미가 다르다.

로그 likelihood는 다음과 같이 분해될 수 있다.

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

- 우리는 autoencoder를 가능성이 높은 $\boldsymbol{h}$ 값으로 point estimate를 해서 위의 식을 근사하는 것으로 생각할 수 있다.

결정된 $\boldsymbol{h}$에 대해, 우리는 다음식을 최적화한다.

$$\log p_{model}(\boldsymbol{h}, \boldsymbol{x}) = \log p_{model}(\boldsymbol{h}) + \log p_{model}(\boldsymbol{x}|\boldsymbol{h})$$

$\log p_{model}(\boldsymbol{h})$항은 sparsity를 초래한다. 예를 들어 라플라스 prior

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

는 절대값 sparsity penalty에 상응한다. log-prior를 절대값 penalty로 표현하면

$$\Omega(\boldsymbol{h}) = \lambda \sum_i |h_i|$$

인데

$$-\log p_{model}(\boldsymbol{h}) = \sum_i (\lambda |h_i| - \log \frac{\lambda}{2}) = \Omega(\boldsymbol{h}) + const$$

$const$ 항은 $\boldsymbol{h}$가 아닌 λ에만 의존한다.

- 우리는 일반적으로 λ는 hyperparameter로 두고, $const$ 항은 패러미터 학습에 영향을 끼치지 않기에 버린다.
- Student-t 같은 다른 prior를 사용해도 sparsity를 초래한다.
- sparsity가 maximum likelihood 학습을 근사하는 과정에서 $p_{model}(\boldsymbol{h})$의 효과로 나온다는 관점에서, sparsity penalty는 regularization 항이 전혀 아니다.
- 그것은 잠재 변수에 대한 모델의 분포의 결과이다.
- 이 관점은 autoencoding 학습의 다른 동기를 제공한다: 그것은 generative model을 근사적으로 학습하는 방법이다.
- 그리고 autoencoder로 학습되는 피쳐가 왜 유용한지를 알려준다: 그것은 input을 설명하는 잠재 변수를 설명해준다.

Autoencoder에 관한 초기 연구는 다양한 sparsity를 탐색하고, sparsity penalty와 undirected probabilistic model $p(\boldsymbol{x}) = \frac{1}{Z}\tilde{p}(\boldsymbol{x})$에 maximum likelihood를 적용하는 과정에서 발생하는 $\log Z$ 항에 관한 연관성을 제안했다.

- 기본 아이디어는 $\log Z$를 최소화하는 것이 확률 모델로 하여금 골고루 흩어지는 것을 방해하고,
- sparsity를 autoencoder에 부여하는 것이, autoencoder로 하여금 모든 것에 low reconstruction error를 가지는 것을 막아준다는 것이다.
- 이 경우는 이 연관은 수학적 대응 보다는 직관적 이해의 레벨이다.
- sparsity penalty에 관한 directed model의 $\log p_{model}(\boldsymbol{h})$에 대응하는 해석은 수학적으로 더 간단하다.

14.2.2 Denoising Autoencoders

- penalty Ω를 코스트 함수에 더하는 대신, 코스트 함수의 reconstruction error 항을 바꾸는 것으로 유용한 것을 배우는 autoencoder를 만들 수 있다.

autoencoder는 아래의 함수를 최적화 한다.

$$L(\boldsymbol{x}, g(f(\boldsymbol{x})))$$

- 여기서 L은 입력값과 다른 $g(f(\boldsymbol{x}))$에 L^2 norm과 같은 penalty를 준다.
- 이것은 $g \circ f$가 capacity만 된다면 identity 함수로 학습되도록 한다.

denoising autoencoder 즉, DAE는 아래를 최소화 한다.

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))$$

여기서 $\tilde{\boldsymbol{x}}$는 $\boldsymbol{x}$의 오염된 복사본이다. 따라서 denoising autoencoder는 단순히 입력을 복사하는 것이 아닌 이 오염을 없에야 한다.

- autoencoder는 reconstruction error를 최소화하는 과정의 결과물이 유용한 속성들을 가질 수 있다는 또 다른 예이다.
- overcomplete하고, high-capacity인 모델도 identity 함수가 되는 것을 주의깊게 관리한다면 autoencoder로 사용될 수 있다는 예이기도 하다.

14.2.3 Regularizing by Penalizing Derivatives

autoencoder를 regularizing하는 다른 방법은 penalty Ω를 sparse autoencoder처럼 사용하되

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}, \boldsymbol{x})$$

다른 형태의 Ω를 사용하는 것이다.

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

- 이렇게 하면 모델은 $\boldsymbol{x}$가 조금 변할 때 많이 변하지 않는 함수를 학습하게 된다.
- 이렇게 regularized된 autoencoder를 **contractive(수축성 있는) autoencoder** 즉 CAE라고 한다.

14.3 Representational Power, Layer Size and Depth

- autoencoder는 주로 단일 레이어 encoder와 단일 레이어 decoder로 학습되곤 한다. 그러나 deep한 encoder / decoder가 갖는 장점이 많다.
- universal approximator theorem에 의해 하나 이상의 hidden layer가 있는 feedforward neural network은, 충분한 hidden unit이 있기만 하면, 어떤 함수도 임의의 정확도로 근사화 할수 있다.

- 그러나, input에서 code로의 매핑은 shallow하기에, code가 sparse해야 한다는 것 같은 임의의 제약을 걸수 없다.
- encoder에 하나 이상의 추가 hidden layer가 있는 deep autoencoder는, 충분한 hidden unit이 있다면, input에서 code로의 매핑을 자유롭게 근사할 수 있다.
- 실험적으로 deep encoder는 shallow나 linear autoencoder보다 훨씬 좋은 효과를 낸다.

- deep autoencoder를 학습시키는 전략은 여러 shallow autoencoder들을 greedy하게 pretrain하는 것이다.

14.4 Stochastic Encoders and Decoders

- Autoencoder도 feedforward network의 일종이다. 전통적인 feedforward network에 사용되던 loss function과 output unit type은 autoencoder에도 그대로 사용될 수 있다.
- 6.2.2.4에서 설명한 것 처럼 feedforward network의 output unit과 loss function의 일반적인 설계 전략은 output distribution $p(\boldsymbol{y}|\boldsymbol{x})$을 정의하고, negative log-likelihood $-\log p(\boldsymbol{y}|\boldsymbol{x})$를 최소화 하는 것이다.

- autoencoder에서 $\boldsymbol{x}$는 traget이자 input이기에, 주어진 hidden code $\boldsymbol{h}$에 대해 decoder는 조건부 분포 $p_{decoder}(\boldsymbol{x}|\boldsymbol{h})$를 제공하는 것으로 생각할 수 있다.
- 이제 $-\log p_{decoder}(\boldsymbol{x}|\boldsymbol{h})$를 최소화 하는 것으로 autoencoder를 학습할 수 있다.

- 일반적인 feedforward network에서는 $\boldsymbol{x}$가 실수면 linear output unit을 사용한다.
- 바이너리 $\boldsymbol{x}$에 대응하는 Bernoulli 분포에 대해서는 softmax distribution을 사용한다.

- 전통적으로 output 변수는 $\boldsymbol{h}$가 주어졌을때 조건부 독립으로 취급되기에 이 확률 분포의 평가는 비싸지 않다.

- 보다 진보적인 시도는 **encoding function** $f(\boldsymbol{x})$을 일반화해 **encoding distribution** $p_{encoder}(\boldsymbol{h}|\boldsymbol{x})$ 개념으로 이해하는 것이다.

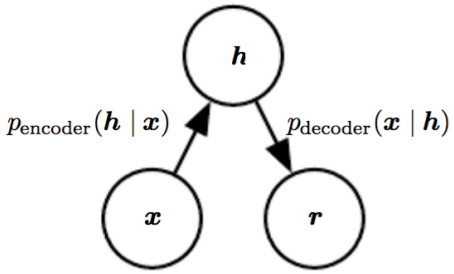


Figure 14.2: The structure of a stochastic autoencoder, in which both the encoder and the decoder are not simple functions but instead involve some noise injection, meaning that their output can be seen as sampled from a distribution, $p_{\text{encoder}}(\boldsymbol{h} \mid \boldsymbol{x})$ for the encoder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

모든 잠재 변수 모델 $p_{model}(\boldsymbol{h}, \boldsymbol{x})$은 stochastic(확률모형) encoder와 decoder를 정의한다.

$$p_{encoder}(\boldsymbol{h}|\boldsymbol{x}) = p_{model}(\boldsymbol{h}|\boldsymbol{x})$$
$$p_{decoder}(\boldsymbol{x}|\boldsymbol{h}) = p_{model}(\boldsymbol{x}|\boldsymbol{h})$$

14.5 Denoising Autoencoders

Denoising autoencoder (DAE)는 오염된 data를 입력으로하여 오염되지 않은 입력을 예측하는 autoencoder이다.

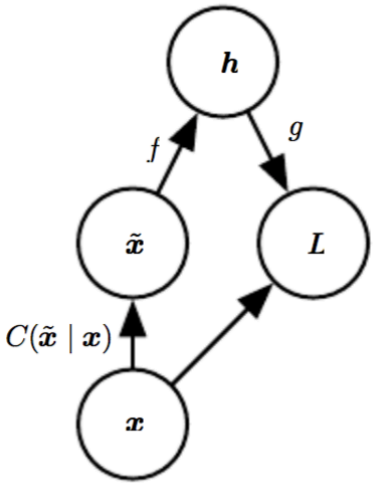


그림 14.3) denoising autoencoder를 위한 코스트 함수의 computational graph. 오염된 데이터 $\tilde{\boldsymbol{x}}$에서 원본 데이터 $\boldsymbol{x}$를 복원하도록 학습된다.

- 로스 $L = -\log p_{decoder}(\boldsymbol{x}|\boldsymbol{h} = f(\tilde{\boldsymbol{x}}))$을 최소화하도록 학습

- $C(\tilde{\mathbf{x}}|\mathbf{x})$는 오염시키는 프로세스 이다.
- 주어진 샘플 $\mathbf{x}$에 대해 오염된 샘플 $\tilde{\mathbf{x}}$의 조건 분포를 나타낸다.

autoencoder는 학습 페어 $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$에서 복원 분포 **reconstruction distribution** $p_{reconstruction}(\mathbf{x}|\tilde{\mathbf{x}})$을 다음과 같은 순서로 추정한다.

1. 학습 데이터에서 $\boldsymbol{x}$를 추출
2. $C(\tilde{\mathbf{x}}|\mathbf{x} = \boldsymbol{x})$에서 오염된 버전인 $\tilde{\boldsymbol{x}}$를 추출
3. $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$을 학습 표본으로 사용해, autoencoder reconstruction 분포 $p_{reconstrunction}(\boldsymbol{x}|\tilde{\boldsymbol{x}}) = p_{decoder}(\boldsymbol{x}|\boldsymbol{h})$를 encoder의 출력 $\boldsymbol{h}$와 $g(\boldsymbol{h})$로 일반적으로 정의되는 $p_{decoder}$ 로 예측.

- 일반적으로 negative log-likelihood, 즉 $-\log p_{decoder}(\boldsymbol{x}|\boldsymbol{h})$에 대해 gradient-based 근사를 수행할 수 있다.
- encoder가 deterministic 하면 denoising autoencoder는 feedforward network이고, 다른 feedforward network과 같은 테크닉으로 학습될 수 있다.

따라서 DAE를 아래와 같은 expectation에 stochastic gradient descent를 적용하는 것으로 볼 수 있다.

$$-\mathbb{E}_{\mathbf{x}\sim\hat{p}_{data}(\boldsymbol{x})}\mathbb{E}_{\tilde{\mathbf{x}}\sim C(\tilde{\mathbf{x}}|\boldsymbol{x})}\log p_{decoder}(\boldsymbol{x}|\boldsymbol{h} = f(\tilde{\boldsymbol{x}}))$$

여기서 $\hat{p}_{data}(\mathbf{x})$는 학습 분포이다.

14.5.1 Estimating the Score

- Score matching은 maximum likelihood의 대안이다.
- 그것은 모델이 모든 학습 샘플 $\boldsymbol{x}$에 대해 같은 score를 가지도록 하여 확률 분포에 관한 일관성 있는 esitimator를 제공한다.

스코어는 특별한 gradient field이다.

$$\nabla_{\boldsymbol{x}}\log p(\boldsymbol{x})$$

- 스코어에 대한 자세한 내용은 18.4에서 다룰 것
- 일단 $\log p_{data}$의 gradient field를 학습하는 것이 그것의 구조를 학습하는 방법 중 하나라는 점을 알아두자.

- DAE의 아주 중요한 속성 중 하나는 그것의 학습 기준(조건부 Gaussian $p(\boldsymbol{x}|\boldsymbol{h})$과 함께)이 autoencoder가 데이터 분포의 스코어를 추정하는 벡터 필드 $(g(f(\boldsymbol{x})) - \boldsymbol{x})$ 를 배우게 한다는 점이다.

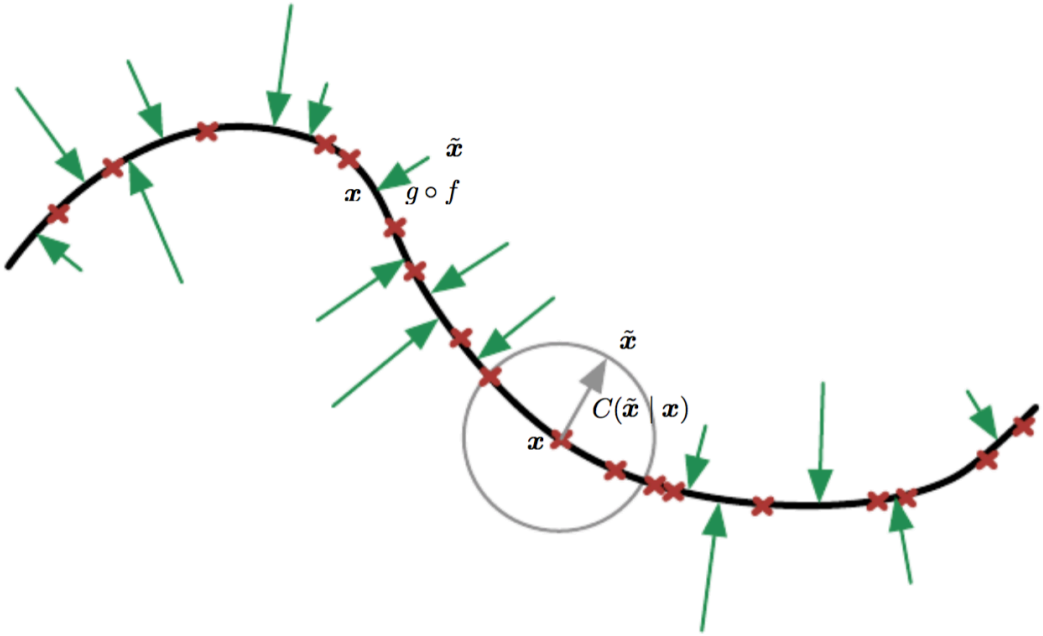


그림 14.4) denoising autoencoder는 오염된 데이터 포인트 $\tilde{\boldsymbol{x}}$을 원래 데이터 포인트 $\boldsymbol{x}$로 복원하도록 학습된다.

- training example $\boldsymbol{x}$를 검은 선으로 표현된 저차원 다양체(manifold) 위의 붉은 화살표로 표시했다.
- 오염 과정 $C(\tilde{\boldsymbol{x}}|\boldsymbol{x})$을 가능한 오염의 회색 원과 함께 표시했다.
- 회색 화살표는 어떤 training example이 이 오염과정을 통해 어떻게 변하는지 보여준다.
- denoising autoencoder가 average squared error $\|g(f(\tilde{\boldsymbol{x}})) - \boldsymbol{x}\|^2$를 최소화하게 학습되었을 때, 복원 $g(f(\tilde{\boldsymbol{x}}))$은 $\mathbb{E}_{\mathbf{x},\tilde{\mathbf{x}} \sim p_{data}(\mathbf{x})C(\tilde{\mathbf{x}}|\mathbf{x})}[\mathbf{x}|\tilde{\mathbf{x}}]$을 추정한다.
- $g(f(\tilde{\boldsymbol{x}}))$가 $\tilde{\boldsymbol{x}}$를 만들었을 수도 있는 원래 포인트 $\boldsymbol{x}$의 무게 중심을 예측하기에, 벡터 $g(f(\tilde{\boldsymbol{x}})) - \tilde{\boldsymbol{x}}$는 manifold의 가장 가까운 점을 근사적으로 향한다.
- 따라서 autoencoder는 녹색 화살표로 표시되는 벡터 $g(f(\boldsymbol{x})) - \boldsymbol{x}$을 배운다.
- 이 벡터는 score $\nabla \log p_{data}(\boldsymbol{x})$을 배수 팩터까지 예측하는데, 그것은 average root mean square 복원 에러이다.

- Gaussian noise와 mean squared error를 reconstruction cost로 사용하여 특정 autoencoder(sigmoidal hidden units, linear reconstruction units)를 denosing 학습하는 것을 Gaussian visible units으로 하는 RBM이라 부른다. (20.5.1에서 설명)
- 현재로는, 그것이 명시적인 $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$을 제공한다는 것만 알면 충분하다.

- RBM이 **denoising score matching**을 이용해 train될 때 학습 알고리즘은 상응하는 autoencoder의 denoising train과 동등하다.
- 노이즈 레벨이 고정되었을 때, regularized score matching은 consistent한 estimator가 아니다; 그것은 그대신 분포의 blurred versions을 복원한다.
- 그렇지만, 샘플 데이터의 수가 무한으로 갈때 noise level이 0으로 가도록 선택되면, consistency는 복원된다.(Denoising score matching에 대해서는 18.5에서 다룸)

- 연속된 값의 $\boldsymbol{x}$에 대해 Gaussian corruption & reconstruction distribution을 가진 denoising 척도(criterion)는 일반적인 encoder와 decoder parameterizations이 적용 가능한 score의 estimator를 낳는다.
- 이것은 squared error criterion를 통한 학습에 의해 score를 추정하도록 만들어진 일반적인 encoder-decoder 아키텍쳐를 의미한다.

squared error criterion은

$$\|g(f(\tilde{\boldsymbol{x}})) - \boldsymbol{x}\|^2$$

이고 corruption은

$$C(\tilde{\mathbf{x}} = \tilde{\boldsymbol{x}}|\boldsymbol{x}) = \mathcal{N}(\tilde{\boldsymbol{x}}; \mu = \boldsymbol{x}, \Sigma = \delta^2 I)$$

이다. 이때 noise variance는 σ^2 이다.

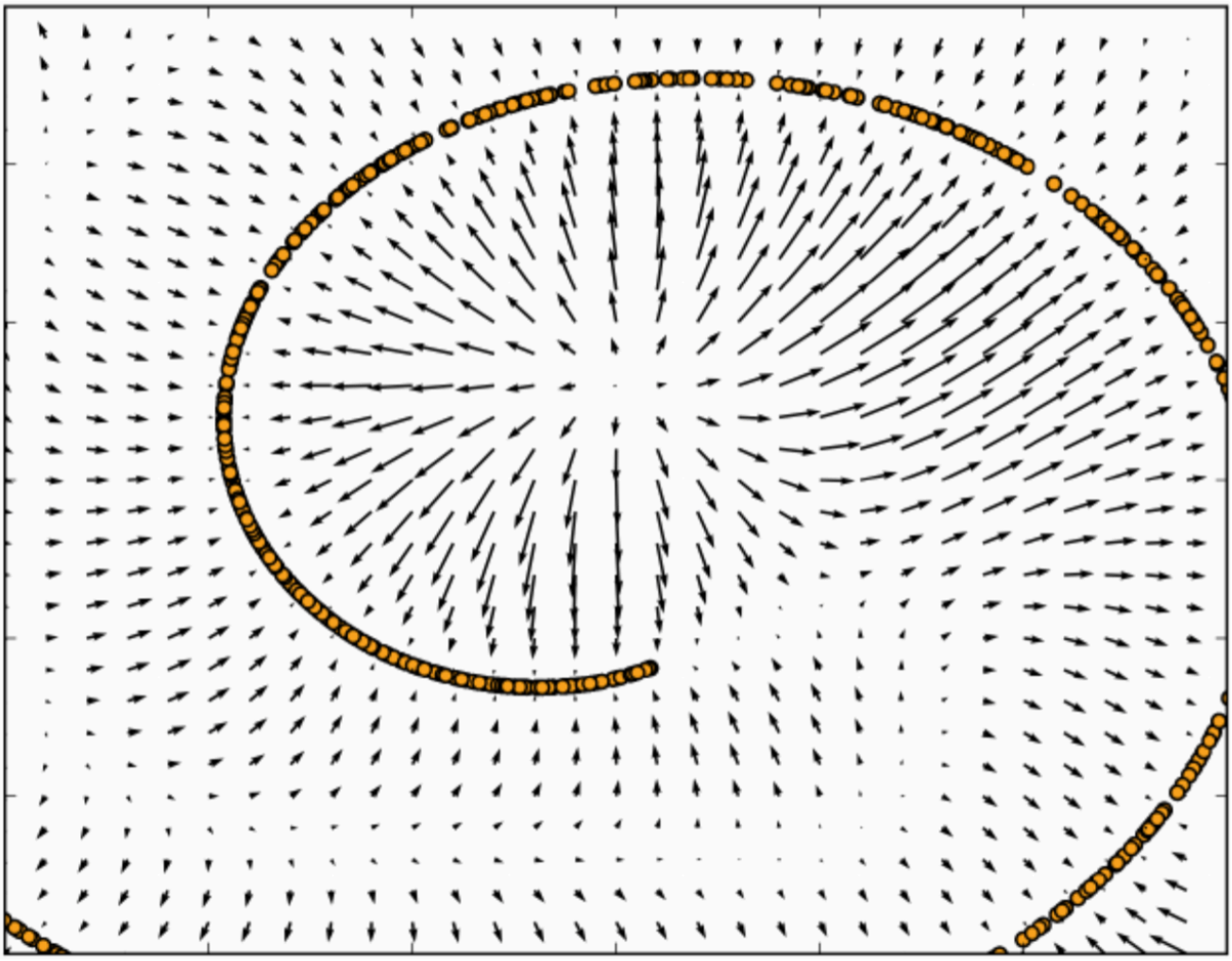


그림 14.5) 2D 공간에서 굽은 1D 다양체(manifold) 주위로 denoising autoencoder를 통해 학습된 벡터 필드

- 각 화살표는 autoencoder의 reconstruction - input 벡터와 묵시적 추정 확률 분포에 의해 더 높은 확률을 향하는 점들에 비례
- 벡터 필드는 추정된 밀도 함수의 최대값과 최소 값에서 0을 가진다.
- 예를 들어, 나선팔은 서로 연결된 local maxima의 1차원 다양체를 형성한다.
- local minima는 두 나선팔의 가운데 갭 근처에 나타난다.
- reconstruction error의 norm(=화살표의 길이로 표시)이 크면, 그 화살표의 방향으로 움직일 때 확률이 크게 증가할 수 있고, 그것은 대부분 확률이 낮은 위치이다.
- autoencoder는 이 확률이 낮은 포인트들을 높은 확률로 복원한다.
- 확률이 최대인 곳에서 복원은 더 정확해 지기에 화살표는 줄어든다.

우리는 지금까지 denosing autoencoder가 어떻게 확률 분포를 represent하는 것을 배우는지 묘사했다. 일반적으로 autoencoder를 generative model로 사용해 이 분포에서 샘플을 뽑기를 원할 수 있다. (20.11에서 다룸)

14.5.1.1 Historical Perspective

- MLP(Multi-Layer Perceptron)을 denoising에 사용하는 아이디어는 1987년 LeCun으로 돌아간다.
- Denoising autoencoder는 어떤 의미에서 단지 denosing하도록 train된 MLP이다.
- 그렇지만 "denoising autoencoder"라는 이름은 denoise 뿐만 아니라 학습과정에서 사이드 이펙트로 좋은 내부 representation을 습득한 모델을 가리킨다.
- 습득된 representation은 더 깊은 자율학습 network 또는 지도학습 network의 pretrain에 사용될 수 있다.

14.6 Learning Manifolds with Autoencoders

- 다른 많은 ML알고리즘 처럼, autoencoder는 낮은 차원의 다양체 또는 그런 다양체의 작은 집합에 데이터가 몰려 있다는 아이디어를 활용한다. (5.11.3)
- 어떤 ML 알고리즘은 이 아이디어를 manifold 위에서는 맞게 동작하는 함수를 학습하는데 활용하지만, 인풋이 manifold를 벗어나면 이상하게 동작한다.
- autoencoder는 이 아이디어를 더 끌고 가서 manifold의 구조를 배우는 것을 목표로 한다.

autoencoder가 이것을 어떻게 하는지 이해하기 위해서 manifold의 중요한 특성을 보아야 한다.

- manifold의 중요한 특성은 그것의 tangent planes 셋이다.
- d 차원의 manifold 상의 점 $\boldsymbol{x}$에서 탄젠트 평면은 manifold 상에서 허용되는 variation의 로컬 방향에 걸치는 d 기저 벡터로 주어진다.
- 아래의 그림처럼, 이 local directions은 $\boldsymbol{x}$를 어떻게 manifold상에 머무르면서도 미량으로 변화시킬 수 있는지 명기한다.

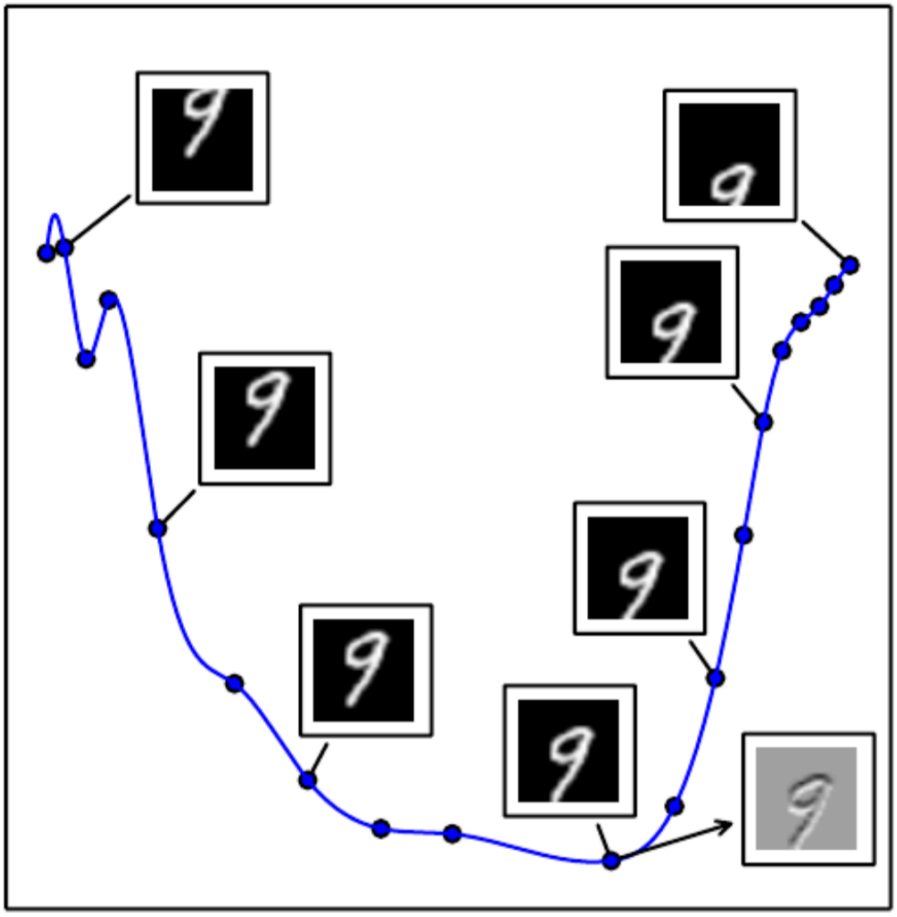


그림 14.6) 탄젠트 하이퍼평면의 개념도

- 784 픽셀로 된 MNIST 이미지를 수직으로 이동하여, 784 차원 공간에 일차원 manifold를 만듦
- 수직 이동의 양은 이미지 공간을 지나는 곡선 경로를 따르는 일차원 다양체를 지나는 어떤 좌표계를 정의한다.
- 위의 그림은 이 다양체를 지나는 몇몇 점들을 보여준다.
- 시각화를 위해 PCA를 통해 manifold를 2차원 공간에 전사했다.
- n차원 다양체는 모든 점에 대해 n차원 탄젠트 평면을 가진다.
- 이 탄젠트 평면은 정확히 그 점에서 만나고, 그 표면에 평행한다.
- 그것은 다양체에 머무른 상태로 움직일 수 있는 공간의 방향을 정의한다.
- 일차원 manifold는 하나의 탄젠트 선을 가진다. 위의 그림에서 탄젠트 방향이 이미지 스페이스에서 어떻게 보이는지를 나타내는 그림을 첨부했다.
- 회색 픽셀은 탄젠트 선을 따라 움직일때 변하지 않는 픽셀이고, 흰 픽셀은 밝아지는 픽셀, 검은 픽셀은 어두워지는 픽셀이다.

모든 autoencoder training 과정은 두 힘간의 타협을 수반한다.

1. training 데이터 $\boldsymbol{x}$가 $\boldsymbol{h}$에서 decoder를 통해서 근사적으로 복원될 수 있게 $\boldsymbol{x}$의 representation $\boldsymbol{h}$를 학습하기.

   $\boldsymbol{x}$가 training data에서 뽑혀 나온다는 것이 아주 중요한데, 그것은 autoencoder가 data generating distribution 아래에서는 있을 수 없는 인풋을 꼭 성공적으로 복원하지는 못한 다는 것을 의미하기 때문이다.
2. 제약조건과 regularization penalty를 만족 시키기.

   이것은 autoencoder의 capacity를 한정하는 구조적인 제약이거나, 아니면 복원 비용에 더해진 regularization 항이 될 수 있다. 이 테크닉은 일반적으로 input에 덜 민감한 해를 찾는다.

- 두 힘들 중 하나만 있으면 유용하지 않다. 두 힘이 함께 있으면 데이터의 생성 분포에 대한 구조를 찾을 수 있어 유용하다.
- 중요한 원칙은 autoencoder는 학습 데이터를 복원하는 데 필요한 variation만 represent할 수 있다는 것이다.
- 만약 데이터 생성 분포가 저차원 다양체 근처에만 집중되어 있으면, 그 다양체의 국지적 좌표계의 representation만 제공할 것이다. 그것은 $\boldsymbol{h} = f(\boldsymbol{x})$의 변화에 대응하기 위한 $\boldsymbol{x}$ 주위 다양체에 대한 variation 탄젠트일 뿐이다.
- 따라서 encoder는 입력 $\boldsymbol{x}$에서 representation space로의 mapping을 배우는데, 그것은 manifold directions 에서 일어나는 변화에만 민감하고 그 다양체에 직교적인 변화에는 반응하지 않는다.

아래는 복원함수를 데이터 포인트 근처의 변화에 무감각하게 하여 autoencoder가 다양체의 구조를 복원하게 하는 것을 보여주는 일차원의 예이다.

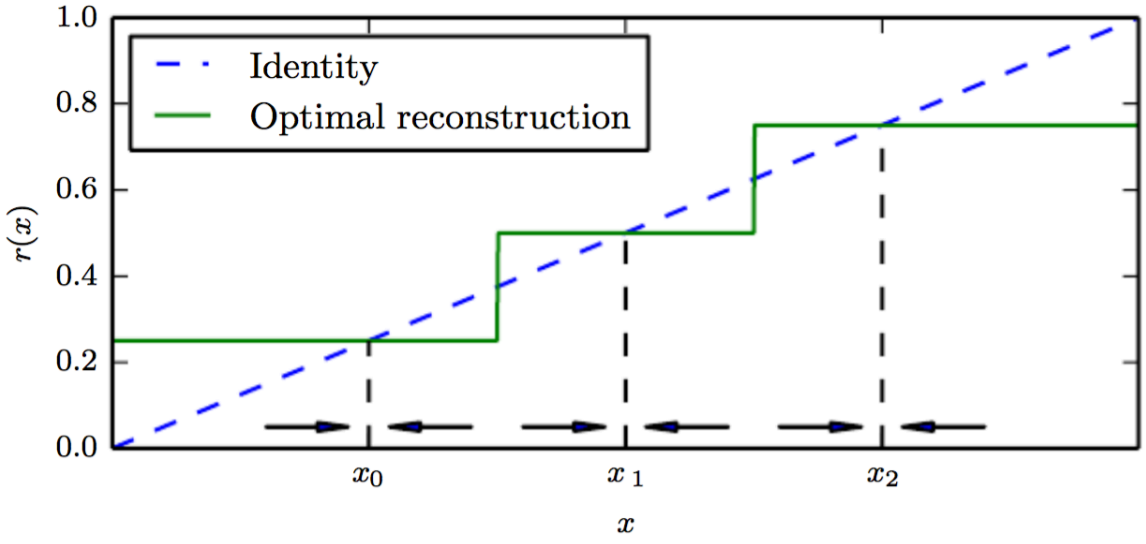


그림 14.7)

- 만약 autoencoder가 data point 근처의 작은 변동에 반응하지 않는 복원함수를 배운다면, 그것은 그 데이터의 manifold structure를 찾아낸 것이다.
- 여기의 다양체 구조는 0 차원 다양체의 집합이다.
- 점사선은 복원함수의 타겟과 같다.
- 최적 복원 함수는 데이터 점이 있는 곳마다 identity function을 지난다.
- 바닥의 수평 화살표들은$r(\boldsymbol{x}) - \boldsymbol{x}$ 즉, 복원 방향 벡터를 나타내는데, 그것은 입력 공간에서, 가장 가까운 "manifold"(1차원의 경우 단일 데이터 포인트)를 가리킨다.
- 비록 데이터 포인트 근처의 $r(\boldsymbol{x})$의 미분값이 작도록 요청되지만, 데이터 포인터들 사이에서는 클 수 있다. 데이터 포인터들 사이 공간은 manifolds 간의 영역에 대응하는데, 이곳은 복원 함수가 오염된 데이터들을 원 manifold로 복원시키기 위해 큰 미분값을 가지는 곳이다.

autoencoder가 manifold learning에 유용한 이유를 이해하기 위해 다른 접근도 살펴보자.

- 다양체를 특징짓기 위해 일반적으로 학습하는 것은 다양체 위의(또는 가까운) 데이터 포인트의 **representation**이다.
- 특정 샘플에 대한 그런 representation을 embedding이라고도 한다.
- 그것은 manifold가 낮은 차원의 subset이 되는 주변 환경 공간보다 작은 차원의 벡터로 표현된다.
- 다른 것들이 주변 공간의 모든 점을 맵핑하는 더 general한 mapping(때로는 encoder라 불리는), 즉 representation function 을 배울 때, 어떤 알고리즘(아래에서 설명할 non-parametric manifold learning)들은 각 학습 데이터에서 직접 embedding(=representation)을 학습한다.

- 다양체 학습은 이 manifold를 찾기 위해 시도하는 자율 학습 과정에 중점을 두고 있다.
- non-linear manifold 학습에 관한 초기 기계학습 연구는 **nearest-neighbor graph**에 기반한 **non-parametric** 방법들에 집중했었다.
- 이 그래프는 학습 샘플 당 하나의 노드와 가까운 이웃을 연결하는 에지들을 가진다.
- 이 방법들은 아래 그림처럼 각 노드를 샘플과 그 이웃간의 차이 벡터와 연결된 variation의 방향이 범위가 되는 탄젠트 평면과 연결한다.

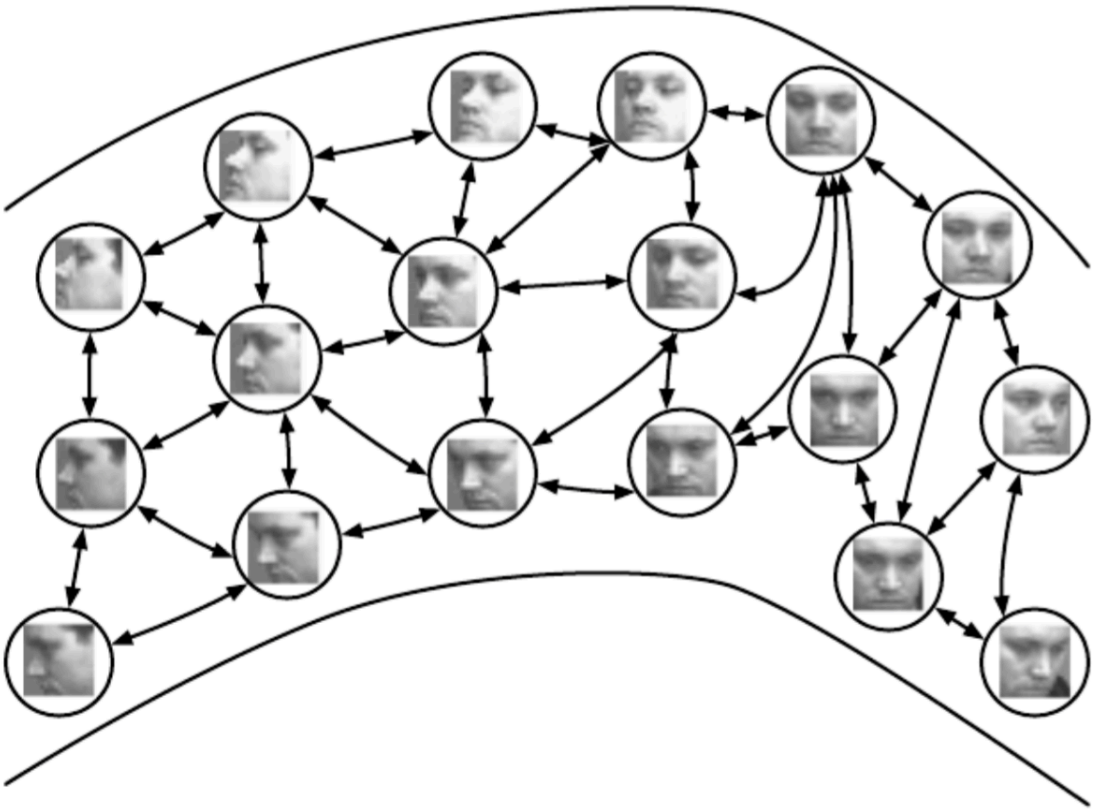


그림 14.8) non-parametric manifold learning 과정은 nearest neighbor graph를 만든다. 거기에서 노드는 각 학습 샘플을 나타내고, 에지는 가장 가까운 이웃과의 관계를 표시한다.

- 다양한 과정을 통해 각 학습 샘플을 실재 값의 벡터 위치, 즉 **embedding**으로 연결해주는 좌표계 뿐만 아니라, 그래프 상의 이웃과 연결된 탄젠트 평면을 구할 수 있다.
- interpolation을 통해 그런 표상을 새로운 데이터를 위해 일반화 하는 것이 가능하다.
- 데이터가 manifold의 굴곡과 뒤틀림을 커버할만큼 충분히 많다면, 이 접근법은 잘 동작한다.

- 이후에 전역 좌표계는 최적화나 linear system을 푸는 것으로 구해질 수 있다.
- 아래의 그림은 국지적으로 linear한 Gaussian-like patch들(가우시안이 탄젠트 방향으로 평평하기에 "pancake"으로도 불림)로 다양체가 어떻게 덮여질 수 있는지 보여준다.

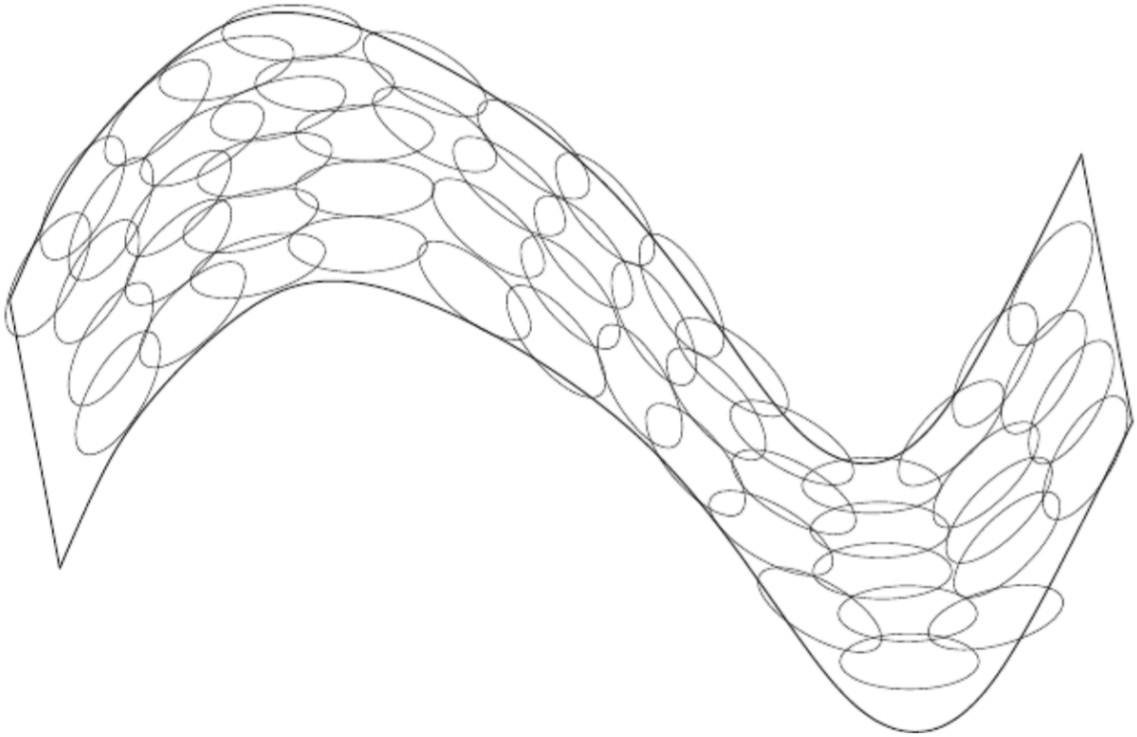


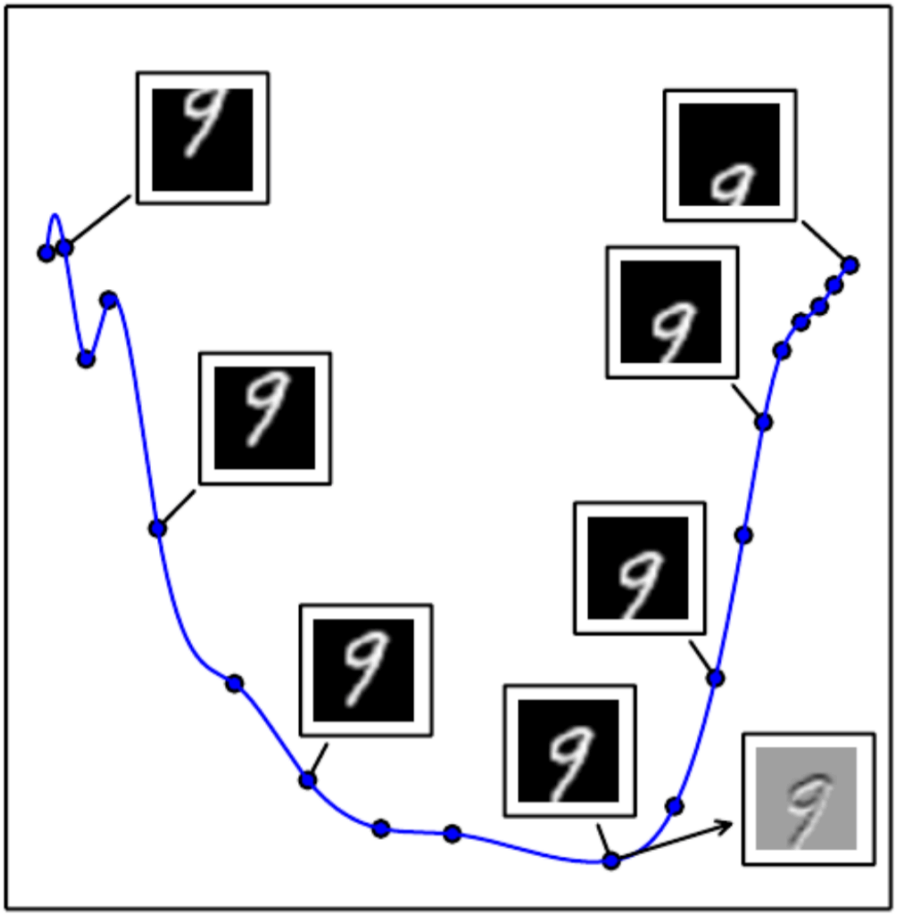
그림 14.9) 만약 각 위치의 탄젠트 평면이 알려져 있다면, 전역 좌표계 또는 density 함수를 구성하기 위해 채워질 수 있다.

- 각 로컬 좌표계는 로컬 Euclidean 좌표계 또는 지역적으로 평평한 Gaussian(팬케익)으로 생각될 수 있는데, 그것은 그 팬케익에 직교하는 방향으로 아주 작은 분산을 가지고, 팬케익 위의 좌표계를 구성하는 방향으로는 큰 분산을 가진다.
- 이 Gaussian들의 조합은 추정된 density function을 제공한다.

하지만, 다양체 학습을 위한 이런 지역적 non-parametric 접근법은 근원적인 어려움이 있다.

- manifold가 부드럽지 않다면(봉우리, 골짜기, 뒤틀린 곳이 많은) 미래의 변동성을 위한 일반화는 차치하고, 이 변동성을 대응하기 위해 아주 많은 학습 데이터가 필요할 수 있다.
- 이 방법은 이웃 데이터간을 보간하는 것으로 manifold의 모습을 일반화 할 수는 있다.
- 그렇지만, AI문제에 등장하는 manifold는 로컬 보간만으로는 파악하기 어려운 복잡한 구조를 가지고 있다.

이동을 통해 나온 manifold 그림 14.6을 다시 생각해보자.



- 입력 벡터 x_i에 대한 하나의 좌표만 본다면, 이미지가 이동하는 것에 따라, 우리는 그 하나의 좌표가 그림 밝기의 모든 봉우리와 골짜기에 대해 그 값의 봉우리 또는 골짜기를 만나는 것을 관측할 것이다.
- 다르게 말하면, 하부 이미지 밝기 패턴의 복잡성이 단순한 이미지 이동으로 생성된 manifold의 복잡성을 초래한다는 것이다.
- 따라서 다양체 구조를 파악을 위해 distributed representation과 딥 러닝을 사용하게 된다.

14.7 Contractive Autoencoders

contractive autoencoder는 코드 $\boldsymbol{h} = f(\boldsymbol{x})$에 대해 f의 미분값이 최대한 작아지게 하는 명시적 regularizer를 도입했다.

$$\Omega(\boldsymbol{x}) = \lambda \left\| \frac{\delta f(\boldsymbol{x})}{\delta \boldsymbol{x}} \right\|_F^2$$

페널티 $\Omega(\boldsymbol{x})$은 encoder 함수에 연결된 편미분 Jacobian matrix의 squared Frobenius norm(=유클리디안 놈, 요소 제곱의 합)이다.

denoising autoencoder와 contractive autoencoder는 관련이 있다:

- 작은 Gaussian input noise의 제약에서, denoising recontruction 에러는 $\boldsymbol{x}$를 $\boldsymbol{r} = g(f(\boldsymbol{x}))$로 매핑하는 복원함수에 대한 contractive 페널티와 같다.
- 다른 말로 하면, denoising autoencoder는 복원 함수를 인풋의 작지만 제한된 크기의 변화에 저항하게 하지만, 수축성있는 autoencoder는 특성 추출 함수가 input의 극미량의 변화에 저항하게 한다.
- 분류기를 위해 피쳐 $f(\boldsymbol{x})$를 사전학습하기 위해 Jacobian 기반의 contractive autoencoder를 사용할 때, 최선의 정확도는 contractive 페널티를 $g(f(\boldsymbol{x}))$가 아닌 $f(\boldsymbol{x})$에 적용하는 것에서 온다.
- $f(\boldsymbol{x})$에 대한 contractive penalty는 14.5.1에서 설명한 것처럼 score matching과도 연관성을 가진다.

- contractive라는 이름은 CAE워프 공간에서 나왔다.
- 분명하게 CAE는 intput의 변동에 저항하도록 훈련되었기에, 입력 포인트의 이웃을 출력 포인트의 더 작은 이웃들로 맵핑하게 한다.
- 우리는 이것을 입력 이웃을 더 작은 출력 이웃으로 수축(contract)하는 것으로 생각할 수 있다.

- 정확히 말하면 CAE는 지역적으로만 contractive하다 - 학습 샘플 $\boldsymbol{x}$의 모든 변동은 $f(\boldsymbol{x})$의 근처에 맵핑된다.
- 전역적으로, 두개의 다른 점 $\boldsymbol{x}$와 $\boldsymbol{x}'$ 은 원래의 점보다 더 멀리 떨어진 $f(\boldsymbol{x})$와 $f(\boldsymbol{x}')$로 맵핑된다.
- f가 데이터 다양체들의 사이 또는 그것들로 부터 멀리 확장된다는 것은 있을 법하다(1-D toy example에서 어떻게 되었는지를 예로 보라).
- $\Omega(\boldsymbol{h})$ 페널티가 시그모이드 유닛들에 적용되었을 때, Jacobian을 쉽게 줄이는 법은 sigmoid 유닛들이 0혹은 1로 포화(saturate)되게 하는 것이다.
- 이것은 CAE가 입력 포인트들을 binary 코드로도 해석될 수 있는 시그모이드의 극단적인 값들로 encode하게 한다.

- 포인트 $\boldsymbol{x}$에서의 Jacobian 행렬 $\boldsymbol{J}$를 non-linear encoder $f(\boldsymbol{x})$를 linear 연산자로 근사하는 것으로 생각할 수 있다.
- 이것은 우리가 수축적(contractive)라는 말을 더 공식적으로 말할 수 있게 해준다.
- 선형 연산자 이론에서 모든 unit-norm $\boldsymbol{x}$에 대해 $\boldsymbol{Jx}$의 놈이 1보다 작거나 같을 때 contractive하다고 말한다.
- 다르게 말하면 $\boldsymbol{J}$는 단위 원을 줄어들게 할때 contractive하다.
- 우리는 CAE를 모든 학습 샘플 $\boldsymbol{x}$에 대해 각 local linear 연산자들을 수축하도록 하기 위해 $f(\boldsymbol{x})$의 로컬 linear 근사의 Frobenius norm을 penalizing하는 것으로 생각할 수 있다.

- 14.6에서 말한 것처럼, regularized autoencoder는 두 대립하는 힘의 균형을 통해 다양체를 학습한다.
- CAE의 경우 이 두힘은 복원 에러와 수축 페널티 $\Omega(\boldsymbol{h})$이다.
- Reconstruction error 만으로는 CAE는 identity function이 될 것이다.
- Contractive penalty 만으로는 CAE는 $\boldsymbol{x}$에 대해 불변한 피쳐를 배울 것이다.
- 이 두 힘간의 타협이 미분값 대부분 경우 $\frac{\delta f(\boldsymbol{x})}{\delta \boldsymbol{x}}$ 이 작은 autoencoder를 생성한다.
- 입력 방향의 작은 수에 대응하는 작은 수의 hidden unit 만이 의미있는 미분값을 가질 수 있다.

- CAE의 목적은 데이터의 manifold 구조를 배우는 것이다.
- 큰 $\boldsymbol{Jx}$를 가지는 방향 $\boldsymbol{x}$는 $\boldsymbol{h}$를 빨리 변화시키기에, 이것들이 manifold의 탄젠트 평면의 근사가 되는 방향일 수 있다.
- 어떤 실험에서 CAE학습의 결과 $\boldsymbol{J}$의 대부분의 singular value는 1이하의 크기로 떨어졌고, 따라서 contractive 해졌다.
- 그렇지만 어떤 singular value들은 1이상에 머물러 있었는데, 그것은 reconstruction error 페널티때문에 CAE가 가장 큰 로컬 분산 방향으로 encode했기 때문이다.
- 가장 큰 singular value와 대응되는 방향들은 수축적 autoencoder가 학습한 탄젠트 평면으로 해석된다.
- 예를 들어 이미지에 대해 적용된 CAE는 14.6에서 보듯, 이미지에 있는 물체의 자세가 점차적으로 변함에 따라 이미지가 어떻게 변하는지를 보여주는 탄젠트 벡터를 학습해야 한다.

아래 그림처럼 실험적으로 얻은 singular 벡터들의 시각화는 입력 이미지의 변환에 의미있게 대응하는 것처럼 보인다.

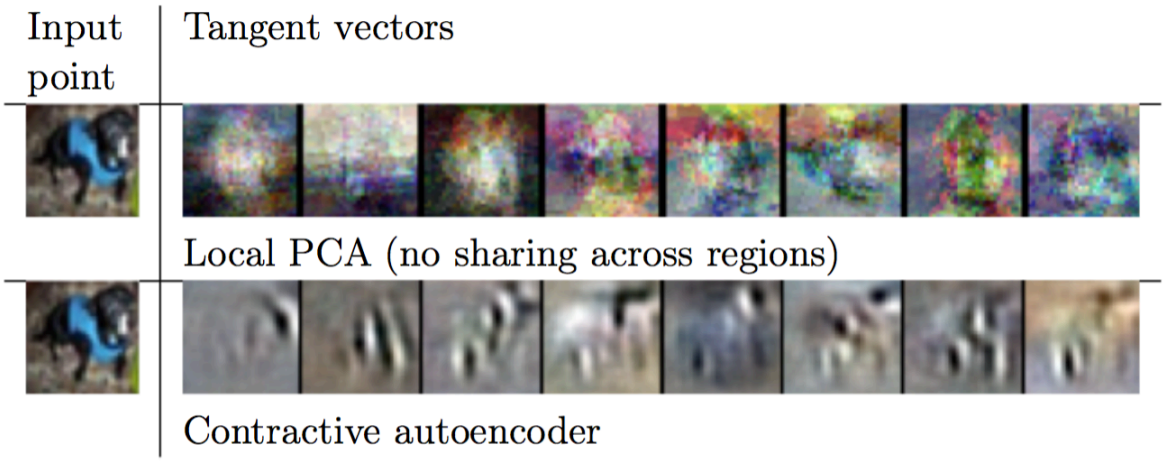


그림 14.10: 로컬 PCA와 contractive autoencoder에 의해 추정된 manifold의 탄젠트 벡터의 모습

- manifold의 위치는 CIFAR-10 데이터 셋에서 가져온 개의 이미지를 입력값으로 해 얻어졌다.
- 탄젠트 벡터는 input-to-code 맵핑의 Jacobian 행렬 $\frac{\delta \boldsymbol{h}}{\delta \boldsymbol{x}}$ 의 leading singular vector로 추정되었다.
- 로컬 PCA와 CAE가 로컬 탄젠트를 찾을 수 있지만, CAE는 활성화된 hidden unit의 subset을 공유하는 서로 다른 위치들 사이에서 패러미터를 공유할 수 있기에 제한된 학습 데이터에서 더 정확한 추정을 할 수 있다.
- CAE 탄젠트 방향은 물체의 움직이거나 변화하는 부분에 대응한다.

- CAE regularization 척도에 관한 실질적인 이슈는 그것이 single hidden layer autoencoder인 경우 계산 비용이 싸지만, 더 깊은 autoencoder의 경우는 훨씬 비싸진다는 점이다.
- 다음의 전략은 single-layer autoencoder들을 따로 학습하고, 이전 autoencoder의 hidden layer를 복원하도록 학습한 후 합성하여 deep autoencoder를 만든다.
- 각 레이어는 지역적으로 contractive하도록 따로 학습되기에, deep autoencoder도 contractive 하다.
- 결과는 전체를 deep model의 Jacobian 페널티로 학습해서 얻어진 것과 같지 않지만, 바람직한 성질을 비슷하게 가지고 있다.

- 다른 실용적인 이슈는 수축 페널티가 우리가 디코더에 어떤 종류의 스케일을 부과하지 않으면 쓸모없는 결과를 얻을 수 있다는 점이다.
- 예를 들어 인코더는 작은 상수 ϵ을 인풋에 곱하는 것으로, 디코더는 코드를 ϵ으로 나누는 것으로 구성될 수 있다.

14.8 Predictive Sparse Decomposition ¶

Predictive sparse decomposition (PSD) 는 sparse coding과 parametric autoencoder의 하이브리드 모델이다.

- parametric encoder는 iterative inference의 출력을 예측하기 위해 학습된다.
- PSD는 오디오 뿐만 아니라 이미지와 비디오 object 인식을 위한 자율 피쳐 학습에 적용되었다.
- 이 모델은 모두 parametric한 encoder $f(\boldsymbol{x})$과 decoder $g(\boldsymbol{h})$로 구성된다.
- 학습 중 $\boldsymbol{h}$는 최적화 알고리즘에 의해 컨트롤 된다. 다음과 같은 식을 최적화 하며 진행된다.

$$\|\boldsymbol{x} - g(\boldsymbol{h})\|^2 + \lambda|\boldsymbol{h}|_1 + \gamma\|\boldsymbol{h} - f(\boldsymbol{x})\|^2$$

- sparse coding과 같이, 학습 알고리즘이 $\boldsymbol{h}$에 대한 최소화와 model 패러미터에 대한 최소화 사이를 결정한다.
- $\boldsymbol{h}$에 대한 최소화는, $f(\boldsymbol{x})$가 $\boldsymbol{h}$의 좋은 초기화 값을 제공하고, cost function이 $\boldsymbol{h}$를 제약해 $f(\boldsymbol{x})$주위에 머무르게 하기에, 빠르다.
- 단순한 gradient descent는 합리적인 $\boldsymbol{h}$값을 10단계 정도의 적은 횟수로 얻을 수 있다.

- PSD가 사용하는 학습 단계는 sparse coding 피쳐의 값을 예측하기 위해 처음에는 sparse coding 모델을 학습하고 다음에 $f(\boldsymbol{x})$를 학습하는 것과는 다르다
- PSD 학습 단계는 $f(\boldsymbol{x})$가 좋은 코드 값을 추론할 수 있는 패러미터를 사용하도록 decoder를 regularize 한다.
- PSD는 19.5에 나올 **learned approximate inference**의 한 예이다.

- PSD의 실용적인 적용에서, 반복적인 최적화는 학습에서만 사용된다.
- parametric encoder f는 모델이 deploy 될 때 학습된 피쳐를 계산하기 위해 사용된다.
- f가 미분가능한 parametric function이기에, f를 계산하는 것은 gradient descent를 통해 $\boldsymbol{h}$을 추론하는 것보다 저렴하다

14.9 Applications of Autoencoders

autoencoder는 공간 축소와 정보 검색에 성공적으로 활용되어 왔다.

참고 RBM) http://deeplearning4j.org/kr-restrictedboltzmannmachine.html (http://deeplearning4j.org/kr-restrictedboltzmannmachine.html)

- 공간 축소는 representation learning과 딥리닝의 최초 응용분야였다.
- Hiton은 RBM의 스택을 학습시켜 그 weight를 hidden layer가 점차적으로 작아지다 30 유닛들의 병목형으로 종결되는 deep autoencoder 초기화에 사용했다.
- 결과 코드는 30차원 PCA보다 복원에러가 적었고, 학습된 표상은 해석하기 쉽고 잘 구분된 클러스터 역할을 하는 기저 카테고리와 연관되어 있었다.

- 저차원의 표상은 classification같은 많은 테스크의 성능을 향상시킬 수 있다.
- 적은 공간을 차지하는 모델은 메모리와 실행시간을 적게 사용한다.
- 차원 축소로 더 많은 해택을 볼 수 있는 분야는 쿼리와 유사한 정보를 database에서 찾는 정보 검색(information retrieval)이다.
- 그것은 차원축소의 혜택뿐만 아니라, 특정 저차원 공간에서 검색이 엄청나게 효과적으로 된다는 장점이 있다.
- 만약 우리가 차원축소 학습을 통해 저차원이고 바이너리한 코드를 생산한다면, 모든 데이터베이스의 항목을 바이너리 코드 벡터에서 항목으로 맵핑하는 해쉬테이블에 저장할 수 있다.
- 이 해쉬테이블을 통해 쿼리와 같은 바이너리 코드를 갖는 모든 데이터베이스 항목을 반환하는 정보 검색을 가능하게 한다.
- 우리는 단지 쿼리의 encoding에서 몇몇 bit를 뒤집는 것으로 약간 덜 닮은 항목도 아주 효율적으로 검색할 수 있다.
- 이렇게 차원 축소와 이진화를 통한 정보 검색법을 **semantic hashing**이라 부르고, 텍스트 입력과 이미지 분야에 적용되고 있다.

- semantic hashing을 위한 이진 코드를 생성하기 위해, 일반적으로 마지막 레이어에 sigmoid 함수를 사용한다.
- 시그모이드 유닛이 거의 0또는 1이 되도록 학습되어야 한다.
- 이것을 간단히 달성하는 방법은 학습중 sigmoid 직전에 추가 노이즈를 끼워 넣는 것이다.
- 노이즈의 크기는 시간이 경과함에 따라 커져야 한다.
- 그 노이즈와 싸워 최대한 정보를 지키기 위해, 네트웍은 포화가 될 때까지 sigmoid의 입력 크기를 키워야한다.

In [ ]: