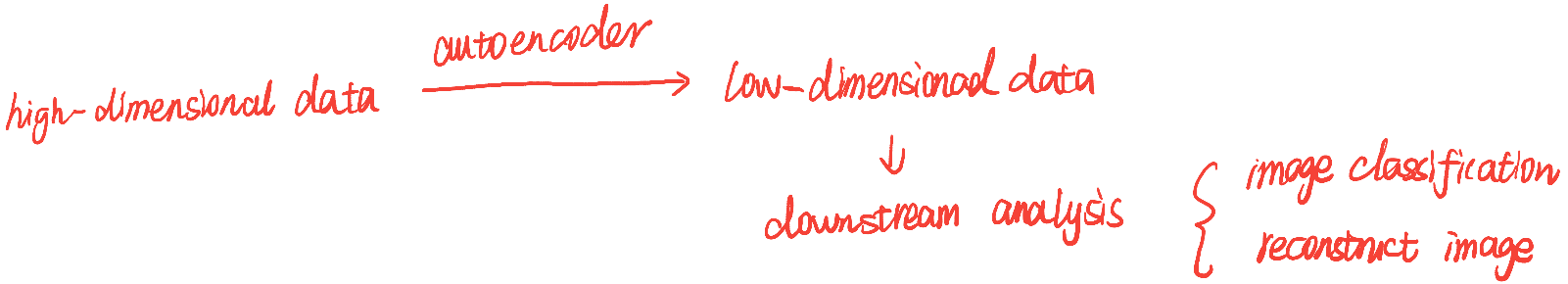
S&DS 365 / 565

Data Mining and Machine Learning

Autoencoders

Yale

Outline



- Autoencoders: Generalization of dictionary learning using deep neural networks
- Variational autoencoders 变体

Minimal neural network: Recall

- Toy data: 3-class spirals
- First discussed a logistic regression model
- Then a simple 2-layer network, backprop calcs

sparse dictionary learning only has 2 layers { input layer / coding layer

These types of networks are sometimes called *multilayer perceptrons*

variational autoencoder : stack layers up

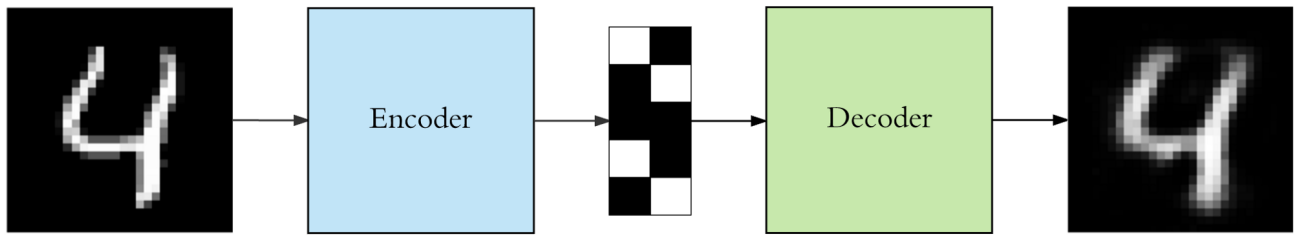
感知机

Autoencoders

- Unsupervised learning methods
- Squeeze high dimensional data through a “bottleneck” of lower dimension
- Train to minimize reconstruction error

Similar to one of interpretation of PCA

Autoencoders



similar to PCA

PCA

Projection Matrix A

Transpose of Projection Matrix A^T

$x \rightarrow [A] \rightarrow Ax \rightarrow A^T(Ax) \rightarrow \hat{x}$

(v)

columns of $A \in \mathbb{R}^{d \times k}$ are orthogonal

https://github.com/ardendertat/Applied-Deep-Learning-with-Keras

Important aspects

goal

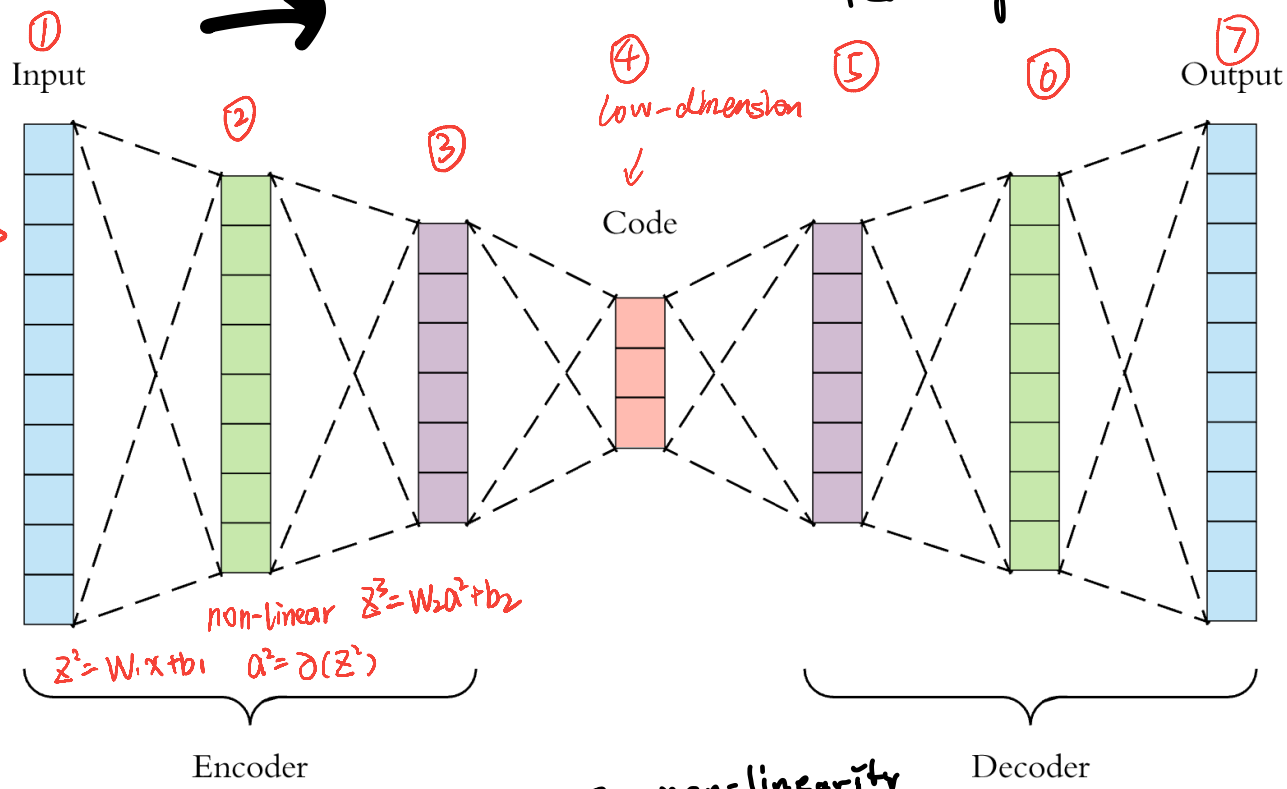
- Unsupervised: No labels used, discovers useful features of input
- Compression: Code reduces dimension of data
- Lossy: Input won't be reconstructed exactly
- Trained: The compression algorithm is learned for specific data

For PCA just use SVD, but can also train as an optimization problem

Deep architecture

similar to Neural Network for supervised learning

Feed Forward: Instead of a label, basically "learning" the input itself!



No hidden layer, all linear

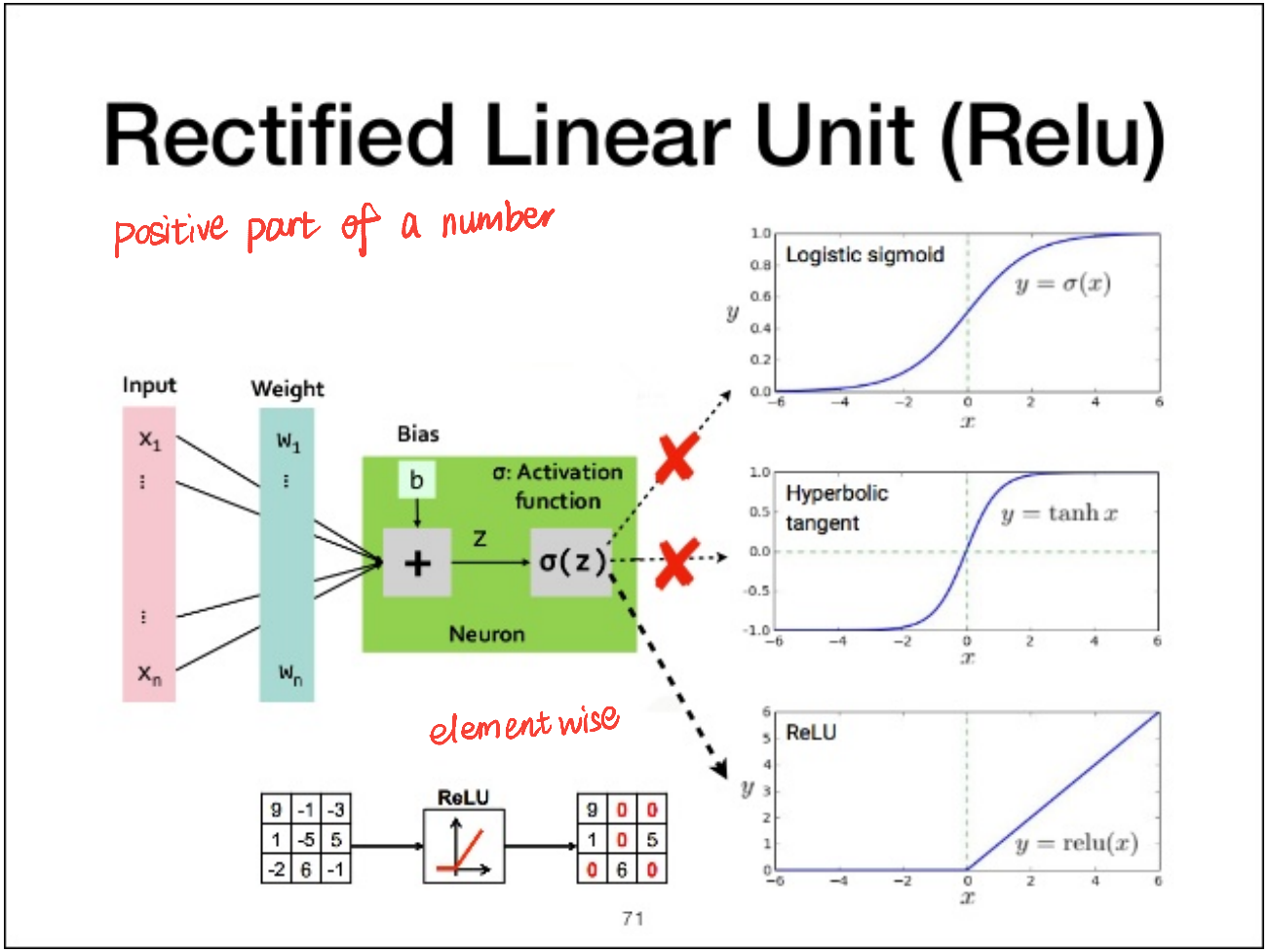
no non-linearity

Pca $x \rightarrow W_1 x \in R^k \rightarrow W_2(W_1 x) \rightarrow \hat{x}$

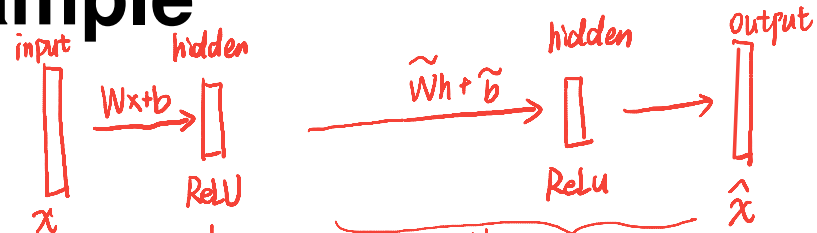
$x \in R^d$ $W_1 \in R^{k \times d}$ $W_2 \in R^{d \times k}$ $\rightarrow \hat{x} \in R^d$

https://github.com/ardendertat/Applied-Deep-Learning-with-Keras

Activation functions



Simple autoencoder example



Encoder network of the form

just one hidden layer

$$h = \mathrm{ReLU}(Wx + b)$$

encoder

where $W \in \mathbb{R}^{H \times D}$ and $b \in \mathbb{R}^H$, decoder network is

$$\hat{x} = \mathrm{ReLU}(\widetilde{W}h + \widetilde{b}) \quad \in R^D$$

where $\widetilde{W} \in \mathbb{R}^{D \times H}$ and $\widetilde{b} \in \mathbb{R}^D$.

Use back-propagation to train this find $W, b, \widetilde{W}, \widetilde{b}$

Objective function: $\text{Loss} = \frac{1}{n}\sum_{i=1}^{n} \|x_i - \hat{x}_i\|_2^2$ reconstruction error = l2-norm squared

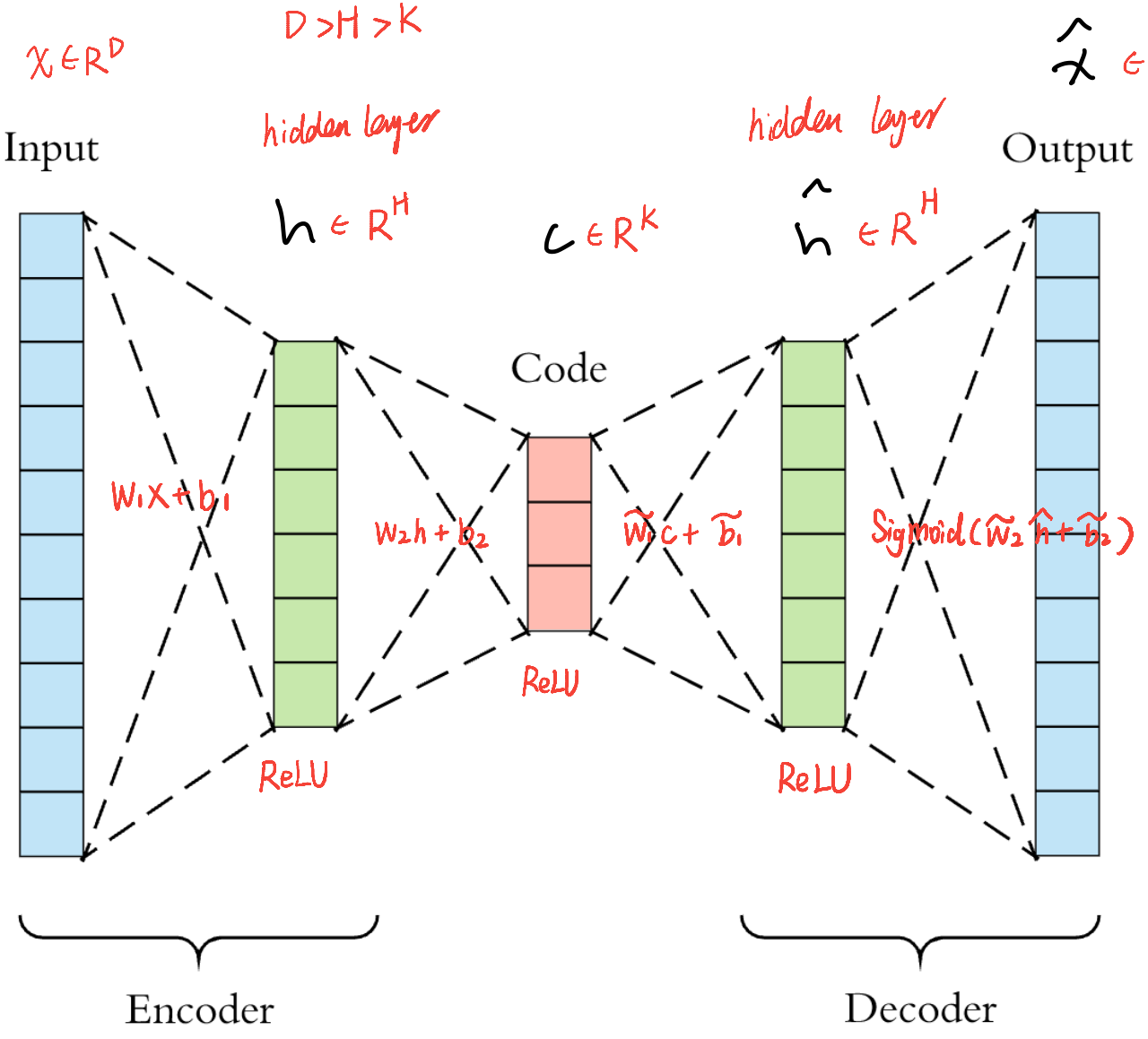
$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \|x_i - \mathrm{ReLU}(\widetilde{W}\,\mathrm{ReLU}(Wx_i + b) + \widetilde{b})\|_2^2.$$

$(+ \lambda \|h_i\|_1)$

$\mathrm{ReLU}(x) = \max(0, x)$, applied component-wise.

element-wise

Simple 2-layer architecture



Simple example – code of dimension K

Encoder network of the form

$$h = \mathrm{ReLU}(W_1 x + b_1) \quad \in R^H$$
$$c = \mathrm{ReLU}(W_2 h + b_2) \quad \in R^K$$

where $W_1 \in \mathbb{R}^{H \times D}$ and $b_1 \in \mathbb{R}^H$, and $W_2 \in \mathbb{R}^{K \times H}$ and $b_2 \in \mathbb{R}^K$

Decoder network of the form

$$\widehat{h} = \mathrm{ReLU}(\widetilde{W}_1 c + \widetilde{b}_1)$$
$$\widehat{x} = \mathrm{Sigmoid}(\widetilde{W}_2 \widehat{h} + \widetilde{b}_2)$$

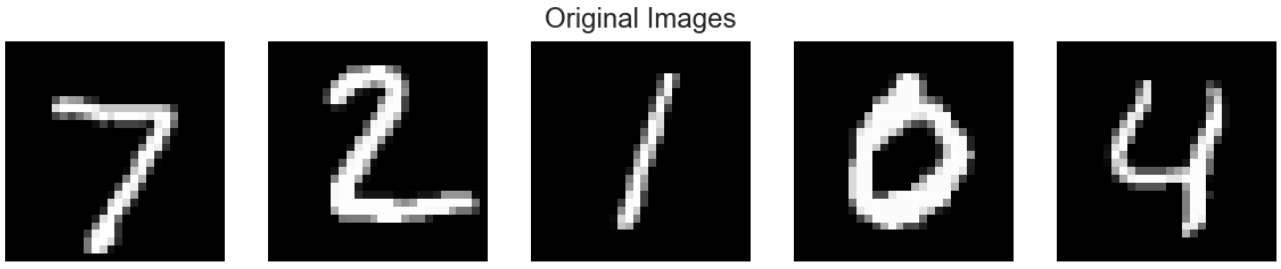
where $\widetilde{W}_1 \in \mathbb{R}^{H \times K}$ and $\widetilde{b}_1 \in \mathbb{R}^H$, and $\widetilde{W}_2 \in \mathbb{R}^{D \times H}$ and $\widetilde{b}_2 \in \mathbb{R}^D$

can use different loss that suit for specific problem

Objective function: binary cross-entropy for 0-1 loss

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \left(x_i \log \widehat{x}_i + (1 - x_i) \log(1 - \widehat{x}_i) \right)$$

MNIST data



Dimension

$$28 \times 28 = 784 = D$$

Adam optimizer

Adaptive gradient descent with momentum

Have package

- Variant of stochastic gradient descent where separate learning rate (step size) is maintained for each network weight (parameter)

different

eg w^1, w^2, b^1, b^2 ...

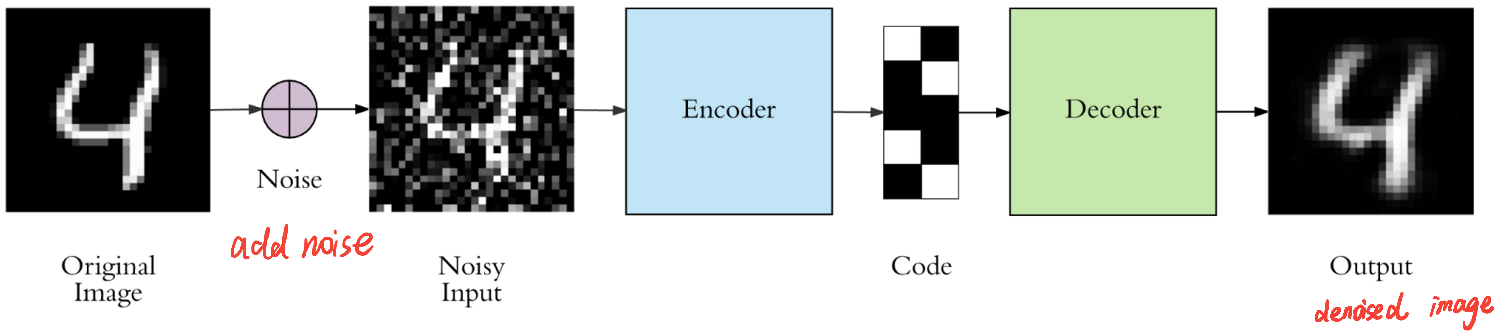
- Each step size adapted as learning progresses based on moments of the derivatives

second

like Hessian

"Adam: A method for stochastic optimization," D. Kingma and J. Ba, https://arxiv.org/abs/1412.6980

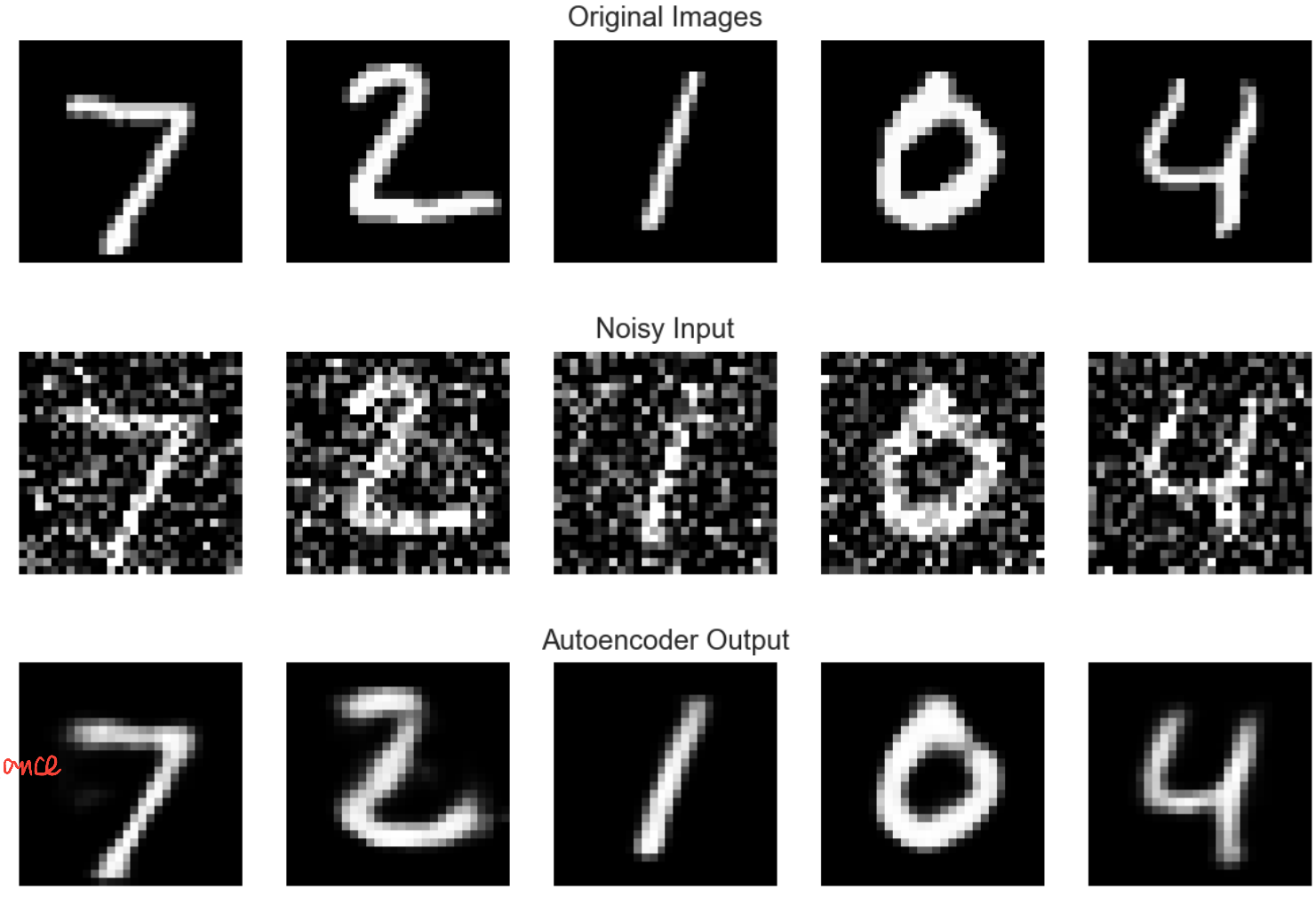
Variant: Denoising autoencoder



- Feed in noisy data
- Train to match to denoised data

minimize reconstruction error between output and original input (not noisy input)

Example result on MNIST



good performance

Variant: Sparse autoencoder

② Loss function: $\mathcal{L} = \mathcal{L} + \lambda \|h_i\|_1$

e.g. add a L-1 norm penalty
want "code" layer to be sparse

$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ ← 1-sparse code

- Add penalty to encourage code sparsity
- Want to find interesting structure
- Generalizes sparse dictionary learning from before

Autoencoders Recap

- Unsupervised learning methods
- Squeeze high-dimensional data through a “bottleneck”
- Train to minimize reconstruction error

Variational autoencoders

- Variational autoencoders offer a more statistical view
- Based on variational inference (方法)
- The decoder is a generative model with a latent variable
- The encoder approximates the posterior distribution

More of Bayesian

for the latent variable

to find the latent variable

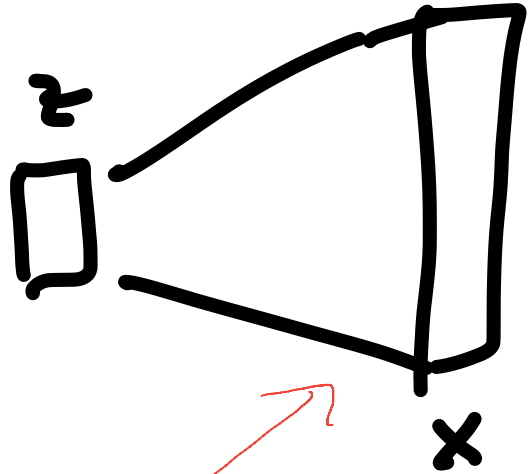
Recap of Bayes:

Latent Random vector Z

$X|Z \sim P(\cdot|Z)$ X is observed

$P(Z|X)$ → inference

Variational autoencoders



Start with a generative model

Suppose a latent variable is normal $Z \sim N(0, I_K)$

$$X \mid Z = z \sim N(G(z), \gamma^2 I_D)$$

$G(z)$ is the *generator network* or *decoder*

As a simple example: One hidden layer

$$G(z) = A_2 \mathrm{ReLU}(A_1 z + b_1) + b_2$$

Transforming Gaussians

Crazy things can happen when you apply activation functions to Gaussians

Transformed Gaussian

仿射 (缩放变换 A + 平移变换 b)

Let $Z \sim N(0, I)$ and an affine mapping

hidden layer

$$h = AZ + b$$

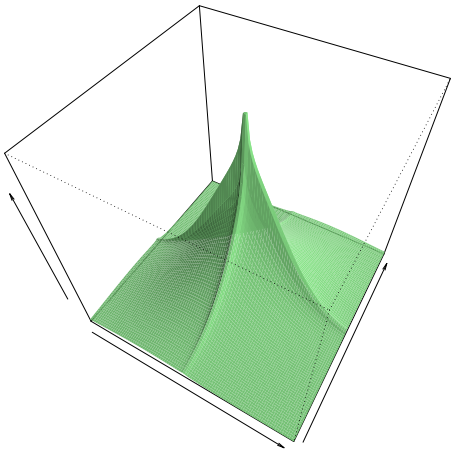
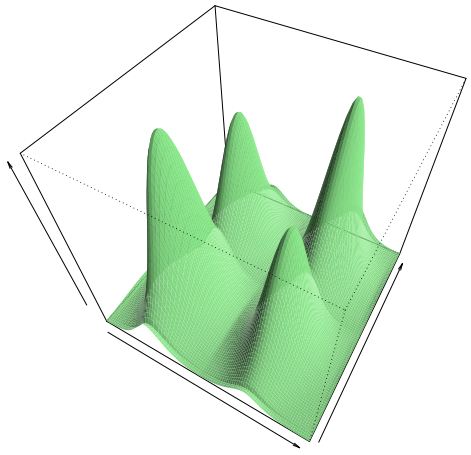
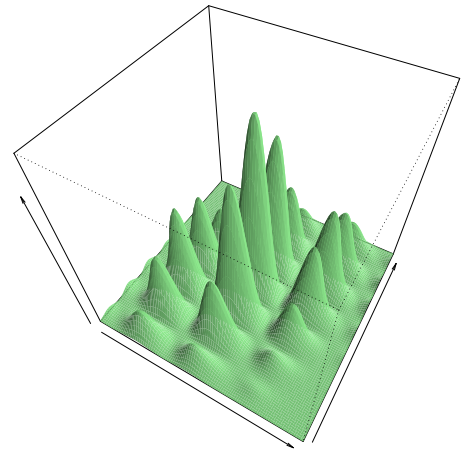
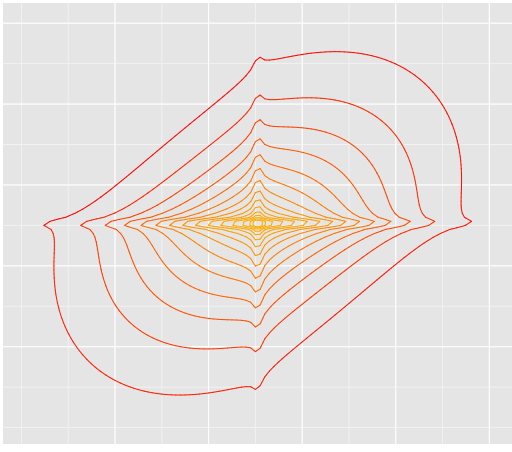
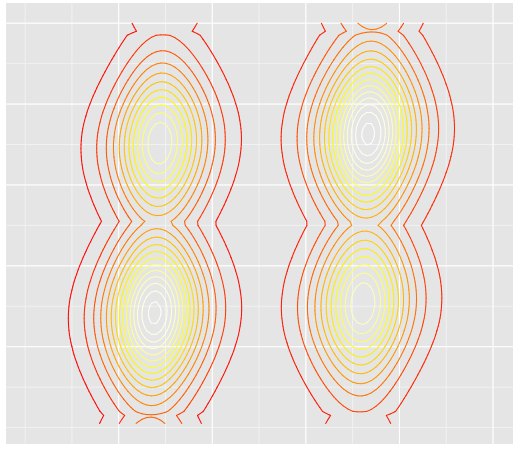
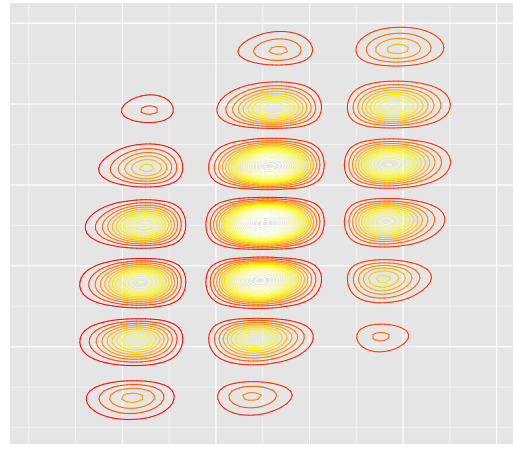
so that $h \sim N(b, AA^T)$.

non-linear 单调变换

Then, apply a monotonic transformation

$$X = g(h)$$

Example densities: $k = 2 \rightarrow d = 2$

Posterior inference

Back to training our generative network:

$$Z \sim N(0, I_K)$$
$$X \mid Z = z \sim N(G(z), \gamma^2 I_D)$$

We want the posterior distribution $p(z \mid x)$.

This is generally hard to compute, because $G(\cdot)$ is nonlinear

Approach: Use variational inference

powful method for solving all Bayesian inference

Mean field variational inference

One method ↑

In mean field variational inference we take

a proposal distribution

$$q(z \mid x) = N(\mu(x), diag(\sigma^2(x)))$$

where now $\mu_j(x)$ and $\sigma_j^2(x)$ are the *variational parameters* for $j = 1 \ldots, K$.

$z \in R^K$ μ and σ^2 are independent

This is a very popular approach for topic models!

Want APPROXIMATE

true posterior distribution $p(z|x)$ with $q(z|x)$! ⇒ find μ and σ^2 that minimize discrepancy between $p(z|x)$ and $q(z|x)$

complex

Using neural networks

Build a "recognition" or encoder neural network that outputs the mean and variance. σ^2 μ

For example: analogous to encoder network of Adam autoencoder encoding x into mean param $\mu(x)$

$$\mu(x) = B_2\,\mathrm{ReLU}(B_1 x + d_1) + d_2$$

and similarly for $\log \sigma^2(x)$. ($\sigma^2 = e^{\log \sigma^2}$)

Using neural networks

For more details David Blei tutorial.

Loss function involves $\mathbb{E}_q(\log p(x|Z))$. Approximate by sampling Z_s for an observed x

$$\mathbb{E}_q(\log p(x|Z)) \approx \frac{1}{N}\sum_{s=1}^{N} \log p(x|Z_s)$$

comes trying to minimize KL divergence between 2 distributions

$\arg\min_{\mu,\sigma^2} KL(q\,||\,p) = \arg\min_q$

key: expectation over q distribution $q(x|z)$ and $p(x|z)$ true

has order, not symmetric : with one takes expectation with respect to

Z is only random variable in equation, x is not random, it's observed

But the parameters of the recognition network have disappeared!

μ, σ

"Reparameterization trick": Re-express the samples Z_s by $Z_i = \mu(x) + \sigma(x)\epsilon_i$ where $\epsilon_i \sim N(0, I_k)$.

D. Kingma and M. Welling, "Autoencoding Variational Bayes," https://arxiv.org/abs/1312.6114. The entropy term isn't a problem since it's just $\log \sigma^2(x)$.

Simple example

Suppose $x \mid z \sim N(G(z), I)$ where generator network is

$$G(z) = \mathrm{ReLU}(Az + b).$$

Negative log conditional probability (NLL?)

Then $-\log p(x \mid z)$ is $\frac{1}{2}$ l_2-norm squared see in linear regression

$\exp\left(-\|x - G(z)\|_2^2\right)$

take -log

$\Rightarrow \|x - G(z)\|_2^2$

$$= \frac{1}{2}\|x - \mathrm{ReLU}(Az + b)\|_2^2$$

Then How to deal with z?

Simple example

Next, suppose the approximate posterior is

$$q(z \mid x) = N(\mu(x), I_K)$$

remove σ^2 in simple e.g means $z = \mu(x) + \epsilon$, $\epsilon \sim N(0, I_K)$

where recognition network is $\mu(x) = \mathrm{ReLU}(Bx + d)$. Then

reparamization trick: instead of randomly sample z, we randomly sample ϵ

For a fixed x, sample a bunch of Z_s (N个) according to distribution q (map): $z = \mu(x) + \epsilon$, $\epsilon \sim N(0, I_k)$

$$-\mathbb{E}_q \log p(x \mid Z) \approx \frac{1}{N} \sum_{s=1}^{N} \frac{1}{2} \|x - \mathrm{ReLU}(AZ_s + b)\|^2$$

sometimes $N = 1$

$= \mathrm{ReLU}(Bx+d) + \epsilon$

notation $\stackrel{d}{=}$ means equal in distribution

$$\stackrel{d}{=} \frac{1}{N} \sum_{s=1}^{N} \frac{1}{2} \|x - \mathrm{ReLU}(A(\mathrm{ReLU}(Bx + d) + \epsilon_s) + b)\|^2$$

ϵ_s in independent of param μ

$\hat{\mu} = \arg\min \left(-\mathbb{E}_{z \sim q_\mu} \log p(x|z)\right)$

train by gradient descent, compute $\frac{\partial E}{\partial \mu}$, but inside of E don't have μ

$$-\mathbb{E}_{z \sim q_\mu} \log(p(x|z)) = \mathbb{E}_{z \sim q} \frac{1}{2} \|x - \mathrm{ReLU}(Az + b)\|^2$$

use reparamization trick, can compute $\frac{\partial \|\|_2^2}{\partial \mu}$, then compute $\frac{\partial E}{\partial e}$ ← 整个等式

$$= \mathbb{E}_{\epsilon \sim N(0, I_K)} \frac{1}{2} \|x - \mathrm{ReLU}(A(\mu(x) + \epsilon) + b)\|^2$$

z 服从 q 分布，q 取决于参数 μ

we want optimize μ

Déjà vu all over again

似曾相识

We saw this before—almost.

Simple autoencoder

$$\|x - \mathrm{ReLU}(A\,\mathrm{ReLU}(Bx + d) + b)\|^2$$

Variational autoencoder Sum over N个 random sample of ϵ_s

$$\frac{1}{N}\sum_{s=1}^{N}\frac{1}{2}\|x - \mathrm{ReLU}(A\,(\mathrm{ReLU}(Bx + d) + \epsilon_s) + b)\|^2$$

① Perspective 1

Similar to simple autoencoder, can think as add noise ϵ_s to code

want reconstruction to be robust against adding noise

Perspective and thinking have changed—back to statistics!

② Perspective 2 : Bayesian

want code have latent variable interpretation: given x, the approximate posterior distribution of z is

Gaussian: $q(z|x) = N(\mu(x), I_k)$

$z = \mathrm{ReLU}(Bx+d) + \epsilon$

Generating data

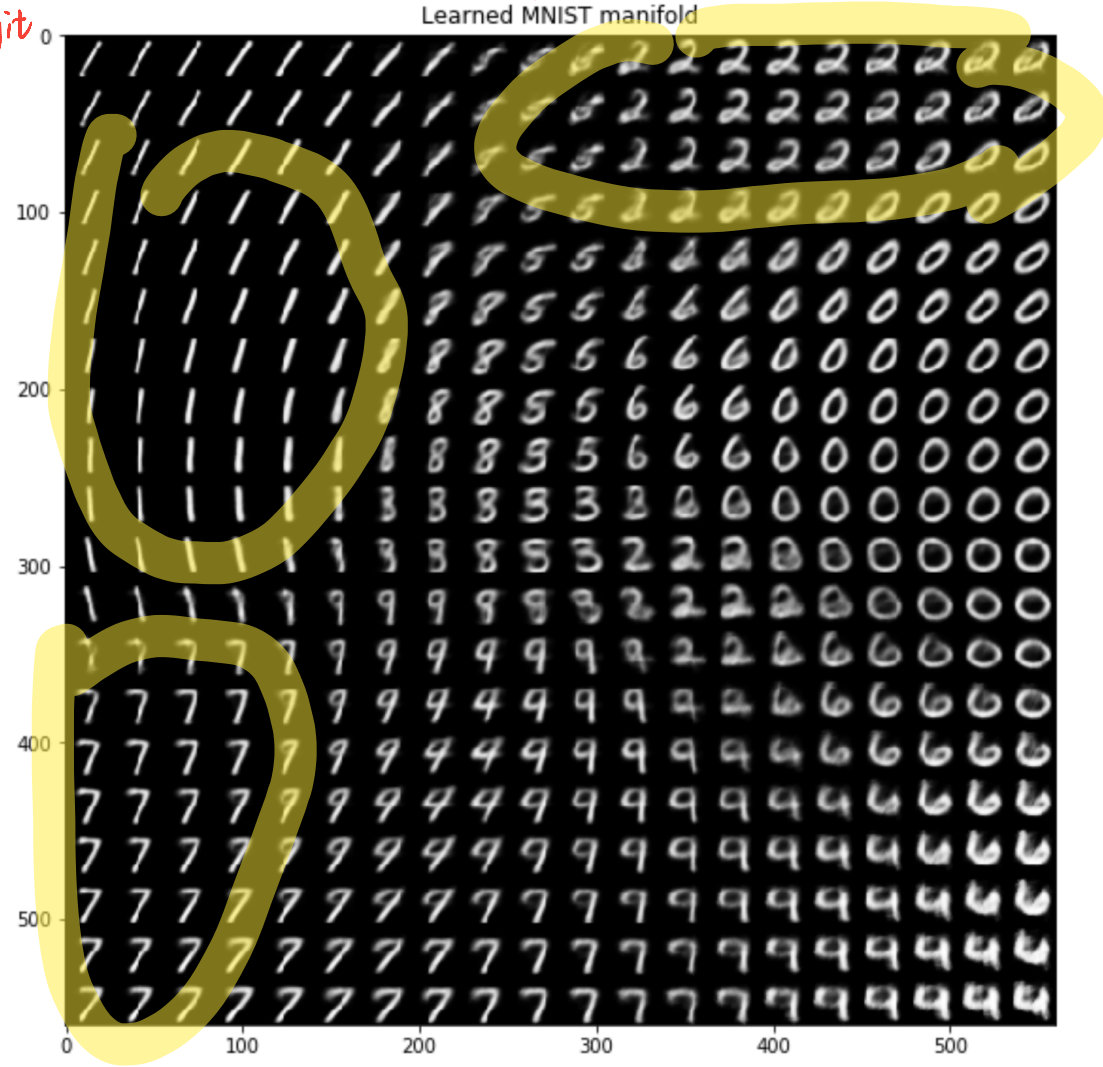
Use generator $G(z) = \text{ReLU}(Az+b)$ to generate data by random zs from Gaussian distribution $z \sim (0, I_k)$

For a tutorial treatment, see https://www.jeremyjordan.me/variational-autoencoders/

Visualizing a 2-dim latent space

2D code

can see transition of digit along 2 directions



For a tutorial treatment, see https://www.jeremyjordan.me/variational-autoencoders/

Interpolating in the latent space

插值

Topic Modelling

Topic Gradation	Generated Equation (Greedily decoded)
Astrophysics (100%)	$G_{\text{eff}} = \frac{1}{2}\left(\frac{M_{\text{eff}}}{M_\odot}\right)^{-1}\left(\frac{M_{\text{eff}}}{M_\odot}\right)^{-1}$
⋮	$G_{\text{eff}} = \frac{1}{2}\left(\frac{M_{\text{eff}}}{M_\odot}\right)^{-1}$
⋮	$G_{\text{eff}} = \frac{1}{2}\left(\frac{1}{2} + \frac{1}{2}\right)$
50% — 50%	$G_{\text{s}} = \frac{1}{2}(1 - \frac{1}{2})$
⋮	$G_{\text{s}}(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{x} + \mathbf{x}^T\mathbf{x}$
⋮	$G_{\text{s}}(\mathcal{C}) = \mathcal{C}(\mathcal{C}) + \mathcal{C}(\mathcal{C}).$
Graph theory (100%)	$G_i = \{(x, y) \in \mathbb{R}^n : x_i = x_i\}$
Optimization (100%)	$L = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_2^2$
⋮	$L = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_2^2$
⋮	$L = \frac{1}{2}\sum_{i=1}^n\sum_{i=1}^n(x_i - x_i)^2 + \sum_{i=1}^n(x_i - x_i)^2$
50% — 50%	$L = \frac{1}{2}\sum_{i=1}^n\sum_{i=1}^n(x_i - x_i)^2 + \sum_{i=1}^n x_i^2$
⋮	$L = \frac{1}{N}\sum_{i=1}^N\sum_{i=1}^N(x_i - x_i)^2$
⋮	$L = \frac{1}{N}\sum_{i=1}^N \mathbb{E}[\mathbf{x}_i^T\mathbf{x}_i],$
Statistics (100%)	$L = \frac{1}{N}\sum_{i=1}^N \mathbb{E}[\mathbf{x}_i^T\mathbf{x}_i]$

Transition

Generator function

Loss function

https://ojs.aaai.org//index.php/AAAI/article/view/4728