



# COMP9444: Neural Networks and Deep Learning

Week 9a. Autoencoders

Alan Blair

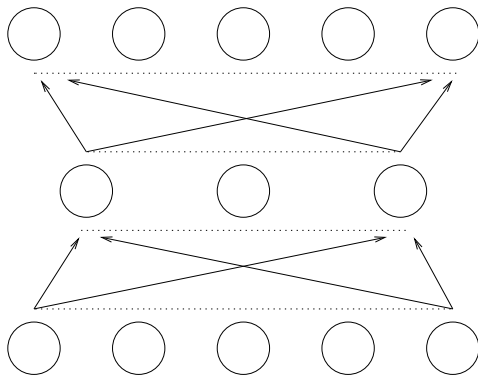
School of Computer Science and Engineering

July 22, 2024

# Outline

- Autoencoder Networks (14.1)
- Regularized Autoencoders (14.2)
- Stochastic Encoders and Decoders (14.4)
- Generative Models
- Variational Autoencoders (20.10.3)

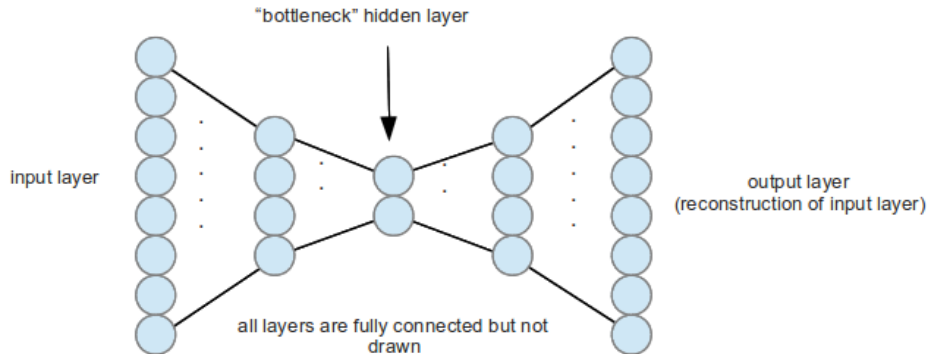
# Recall: Encoder Networks



Inputs	Outputs
10000	10000
01000	01000
00100	00100
00010	00010
00001	00001

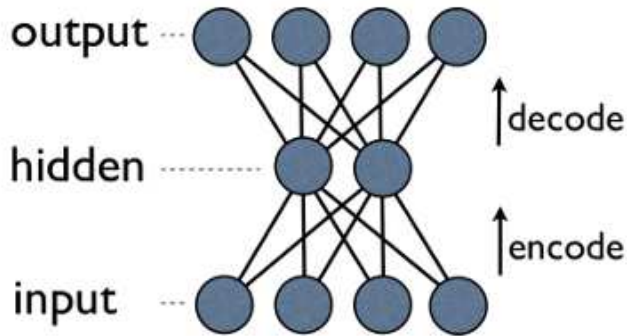
- identity mapping through a bottleneck
- also called N–M–N task
- used to investigate hidden unit representations

# Autoencoder Networks



- ➔ output is trained to reproduce the input as closely as possible
- ➔ activations normally pass through a bottleneck, so the network is forced to compress the data in some way
- ➔ Autoencoders can be used to generate “fake” items, or to automatically extract abstract features from the input

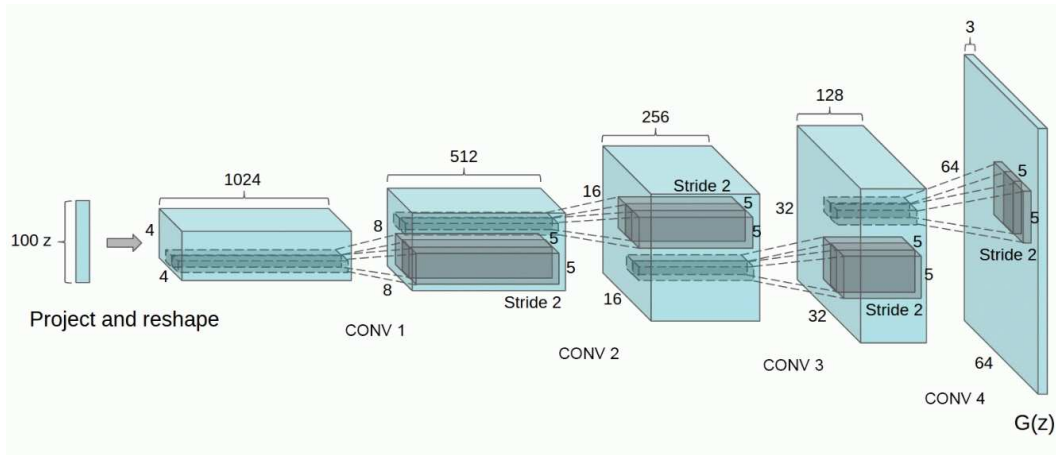
# Autoencoder Networks



If the encoder computes  $z = f(x)$  and the decoder computes  $g(f(x))$  then we aim to minimize some distance function between  $x$  and  $g(f(x))$

$$E = L(x, g(f(x)))$$

# De-Convolutional Encoder for Images



Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks (Radford et al., 2016)

# Autoencoder as Pretraining

- after an autoencoder is trained, the decoder part can be removed and replaced with, for example, a classification layer
- this new network can then be trained by backpropagation
- the features learned by the autoencoder then serve as initial weights for the supervised learning task

## Regularized Autoencoders (14.2)

We may include additional loss term(s) in order to force the latent variables to conform to a certain distribution, or to achieve some other objective.

- Autoencoders with dropout at hidden layer(s)
- Sparse Autoencoders
- Contractive Autoencoders
- Denoising Autoencoders
- Variational Autoencoders
- Wasserstein Autoencoders



## Sparse Autoencoder (14.2.1)

- One way to regularize an autoencoder is to include a penalty term in the loss function, based on the hidden unit activations.
- This is analagous to the weight decay term we previously used for supervised learning.
- One popular choice is to penalize the sum of the absolute values of the activations in the hidden layer

$$E = L(x, g(f(x))) + \lambda \sum_i |h_i|$$

- This is sometimes known as  $L_1$ -regularization (because it involves the absolute value rather than the square); it can encourage some of the hidden units to go to zero, thus producing a sparse representation.

## Contractive Autoencoder (14.2.3)

- Another popular penalty term is the  $L_2$ -norm of the derivatives of the hidden units with respect to the inputs

$$E = L(x, g(f(x))) + \lambda \sum_i ||\nabla_x h_i||^2$$

- This forces the model to learn hidden features that do not change much when the training inputs  $x$  are slightly altered.

## Denoising Autoencoder (14.2.2)

Another regularization method, similar to contractive autoencoder, is to add noise to the inputs, but train the network to recover the original input

*repeat:*

*sample a training item  $x^{(i)}$*

*generate a corrupted version  $\tilde{x}$  of  $x^{(i)}$*

*train to reduce  $E = L(x^{(i)}, g(f(\tilde{x})))$*

*end*

# Loss Functions and Probability

- We saw previously how the loss (cost) function at the output of a feedforward neural network (with parameters  $\theta$ ) can be seen as defining a probability distribution  $p_{\theta}(x)$  over the outputs. We then train to maximize the log of the probability of the target values.
  - squared error assumes an underlying Gaussian distribution, whose mean is the output of the network
  - cross entropy assumes a Bernoulli distribution, with probability equal to the output of the network
  - softmax assumes a Boltzmann distribution

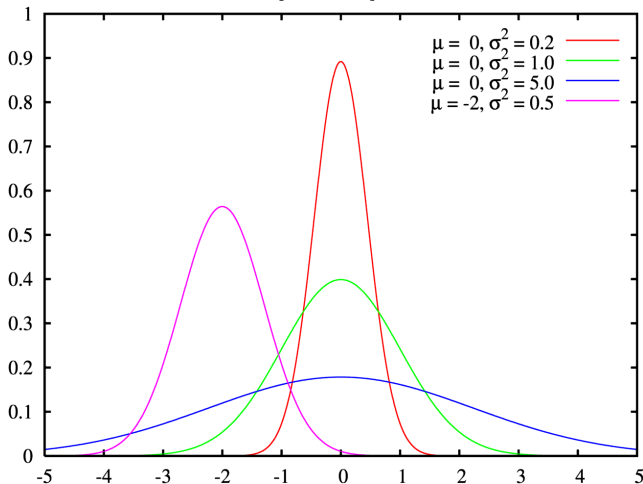
# Stochastic Encoders and Decoders (14.4)

- For autoencoders, the decoder can be seen as defining a conditional probability distribution  $p_{\theta}(x|z)$  of output  $x$  for a certain value  $z$  of the hidden or “latent” variables.
- In some cases, the encoder can also be seen as defining a conditional probability distribution  $q_{\phi}(z|x)$  of latent variables  $z$  based on an input  $x$ .

# Generative Models

- Sometimes, as well as reproducing the training items  $\{x^{(i)}\}$ , we also want to be able to use the decoder to generate new items which are of a similar “style” to the training items.
- In other words, we want to be able to choose latent variables  $z$  from a standard Normal distribution  $p(z)$ , feed these values of  $z$  to the decoder, and have it produce a new item  $x$  which is somehow similar to the training items.
- Generative models can be:
  - explicit (Variational Autoencoders, Wasserstein Autoencoders)
  - implicit (Generative Adversarial Networks)

# Gaussian Distribution (3.9.3)



$\mu$  = mean

$\sigma$  = standard deviation

$$P_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$

# Entropy and KL-Divergence

- The *entropy* of a distribution  $q()$  is 
$$H(q) = \int_{\theta} q(\theta)(-\log q(\theta))d\theta$$
- In Information Theory,  $H(q)$  is the amount of information (bits) required to transmit a random sample from distribution  $q()$
- For a Gaussian distribution, 
$$H(q) = \sum_i \log \sigma_i$$
- KL-Divergence 
$$D_{\text{KL}}(q \parallel p) = \int_{\theta} q(\theta)(\log q(\theta) - \log p(\theta))d\theta$$
- $D_{\text{KL}}(q \parallel p)$  is the number of *extra* bits we need to transmit if we designed a code for  $p()$  but the samples are drawn from  $q()$  instead.
- If  $p(z)$  is Standard Normal distribution, minimizing  $D_{\text{KL}}(q_{\phi}(z) \parallel p(z))$  encourages  $q_{\phi}()$  to center on zero and spread out to approximate  $p()$ .



# KL-Divergence and Wasserstein Distance

Consider two Gaussian distributions  $q, p$  with mean  $\mu_1, \mu_2$  and covariance  $\Sigma_1, \Sigma_2$ , respectively. In the case where  $\mu_2 = 0$ ,  $\Sigma_2 = I$ , the KL-Divergence between  $q$  and  $p$  simplifies to:

$$D_{\text{KL}}(q||p) = \frac{1}{2} [||\mu_1||^2 + \text{Trace}(\Sigma_1) - \log |\Sigma_1| - d]$$

If  $\Sigma_1 = \text{diag}(\sigma_1^2, \dots, \sigma_d^2)$  is diagonal, this further simplifies to:

$$D_{\text{KL}}(q||p) = \frac{1}{2} [||\mu_1||^2 + \sum_{i=1}^d (\sigma_i^2 - 2 \log(\sigma_i) - 1)]$$

which is minimized when  $\mu_1 = 0$  and  $\sigma_i = 1$  for all  $i$ .

The *Wasserstein Distance* between  $q$  and  $p$  is given by

$$W_2(q, p)^2 = ||\mu_1 - \mu_2||^2 + \text{Trace}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1 \Sigma_2)^{\frac{1}{2}})$$

## Variational Autoencoder (20.10.3)

Instead of producing a single  $z$  for each  $x^{(i)}$ , the encoder (with parameters  $\phi$ ) can be made to produce a mean  $\mu_{z|x^{(i)}}$  and standard deviation  $\sigma_{z|x^{(i)}}$

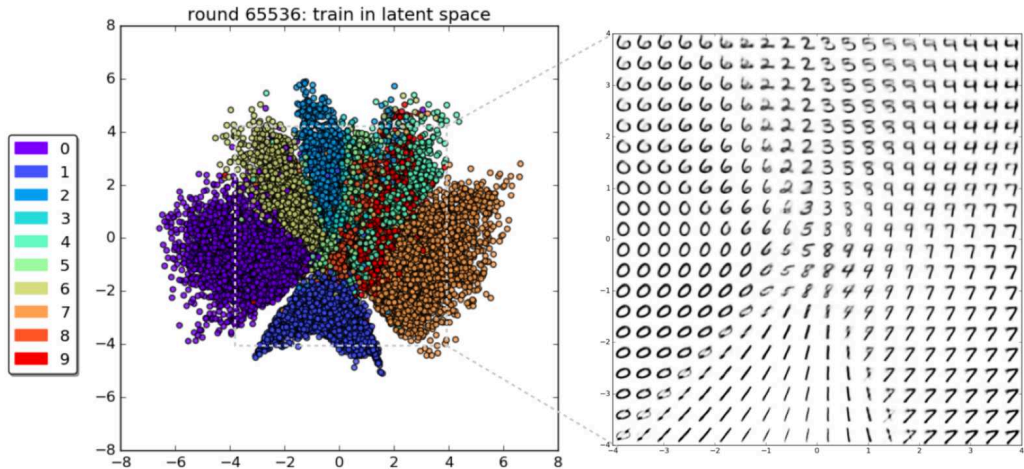
This defines a conditional (Gaussian) probability distribution  $q_{\phi}(z|x^{(i)})$

We then train the system to maximize

$$\mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} [\log p_{\theta}(x^{(i)}|z)] - D_{\text{KL}}(q_{\phi}(z|x^{(i)}) \| p(z))$$

- the first term enforces that any sample  $z$  drawn from the conditional distribution  $q_{\phi}(z|x^{(i)})$  should, when fed to the decoder, produce something approximating  $x^{(i)}$
- the second term encourages  $q_{\phi}(z|x^{(i)})$  to approximate  $p(z)$
- in practice, the distributions  $q_{\phi}(z|x^{(i)})$  for various  $x^{(i)}$  will occupy complementary regions within the overall distribution  $p(z)$

# Variational Autoencoder Digits



# Variational Autoencoder Digits



1st Epoch



9th Epoch



Original

# Variational Autoencoder Faces



# Variational Autoencoder

- Variational Autoencoder produces reasonable results
- tends to produce blurry images
- often end up using only a small number of the dimensions available to  $z$
- another model, called *Wasserstein Autoencoder*, aims to minimize the Wasserstein distance rather than the KL-Divergence

# References

<http://kvfrans.com/variational-autoencoders-explained/>

Carl Doersch, “Tutorial on Variational Autoencoders”.

<https://arxiv.org/pdf/1606.05908.pdf>

I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schölkopf, “Wasserstein Auto-Encoders”, in ICLR, 2018. <https://arxiv.org/pdf/1711.01558>