Deep Learning

7. Autoencoders

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

Tue. 21.4.	(1)	1. Supervised Learning (Review 1)
Tue. 28.4.	(2)	2. Neural Networks (Review 2)
Tue. 5.5.	(3)	3. Regularization for Deep Learning
Tue. 12.5.	(4)	4. Optimization for Training Deep Models
Tue. 19.5.	(5)	5. Convolutional Neural Networks
Tue. 26.5.	(6)	6. Recurrent Neural Networks
Tue. 2.6.	—	— *Pentecoste Break* —
Tue. 9.6.	(7)	7. Autoencoders
Tue. 16.6.	(8)	8. Generative Adversarial Networks
Tue. 23.6.	(9)	9. Recent Advances
Tue. 30.6.	(10)	10. Engineering Deep Learning Models
Tue. 7.7.	(11)	tbd.
Tue. 14.7.	(12)	Q & A

Outline

1. Dimensionality Reduction and Semi-Supervised Learning (Review)
2. Autoencoders
3. Layers Parametrizing Distributions
4. Variational Autoencoders

Outline

Dimensionality Reduction / Via Feature Reconstruction

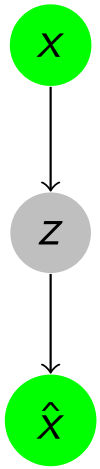
Given a dataset $\mathcal{D} = (x_1, x_2, \ldots, x_N) \in (\mathbb{R}^M)^*$,
an **embedding dimension** $K \in \mathbb{N}$, and
a pairwise loss $\ell : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$,

find an **embedding** / **encoder**

$$z : \mathbb{R}^M \to \mathbb{R}^K$$

and a **reconstruction map** / **(auto)decoder**

$$\hat{x} : \mathbb{R}^K \to \mathbb{R}^M$$



with minimal reconstruction error

$$\ell(z, r; \mathcal{D}) := \frac{1}{N} \sum_{n=1}^{N} \ell(x_n, \hat{x}(z(x_n)))$$

- $K \ll M$
  - otherwise there is a trivial solution $z(x) := x, \hat{x}(z) = z$ with error 0.
- z is called **lower-dimensional / latent representation**.

Linear Dimensionality Reduction / Linear Autoencoders

- restrict encoder z and decoder $\hat{x}$ to be linear:

$$z(x) := Wx, \quad \hat{x}(z) = Vz, \quad W \in \mathbb{R}^{K \times M}, V \in \mathbb{R}^{M \times K}$$
$$\rightsquigarrow \quad \hat{x}(x) = VWx$$

Q: Are linear autoencoders useful?

Linear Dimensionality Reduction / Linear Autoencoders

- restrict encoder z and decoder $\hat{x}$ to be linear:

$$z(x) := Wx, \quad \hat{x}(z) = Vz, \quad W \in \mathbb{R}^{K \times M}, V \in \mathbb{R}^{M \times K}$$
$$\rightsquigarrow \quad \hat{x}(x) = VWx$$

  - view $x_k^{\text{proto}} := V_{1:M,k}$ as an instance prototype.
  - every instance is reconstructed as linerar combinaton of these prototypes:

$$\hat{x} = \sum_{k=1}^{K} z_k x_k^{\text{proto}}$$

- for $\ell(x, \hat{x}) = ||x - \hat{x}||_2^2$: **principle components analysis** (PCA)

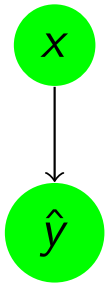
What is Dimensionality Reduction Useful For?

- to visualize data ($K = 2$ or $K = 3$)
- to compress data for transmitting or storage
- for feature compression instead of feature selection
- for semi-supervised learning

Semi-supervised Learning

Given a labeled dataset $\mathcal{D}^{\text{train}} = ((x_1, y_1), \ldots, (x_N, y_N)) \in (\mathbb{R}^M \times \mathbb{R}^O)^*$,
an **unlabeled dataset** $\mathcal{D}^{\text{unlab}} = (x'_1, x'_2, \ldots, x'_{N'}) \in (\mathbb{R}^M)^*$,
a pairwise loss $\ell : \mathbb{R}^O \times \mathbb{R}^O \to \mathbb{R}$,

find a **model**



$$\hat{y} : \mathbb{R}^M \to \mathbb{R}^O$$

with minimal error on fresh data (from the same distribution):

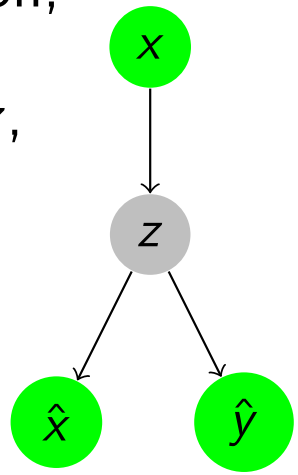
$$\ell(\hat{y}; \mathcal{D}^{\text{test}}) := \frac{1}{|\mathcal{D}^{\text{test}}|} \sum_{(x,y) \in \mathcal{D}^{\text{test}}} \ell(y, \hat{y}(x))$$

- the unlabeled data is usually way larger than the labeled on.

Q: what is the unlabeled data useful for?

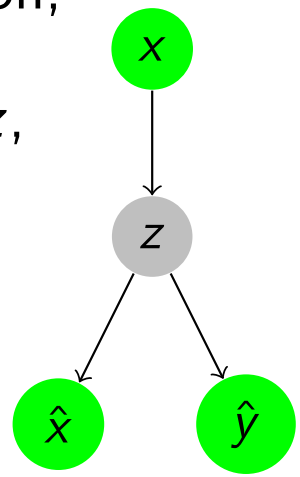
Semi-supervised Learning / Using the Unlabeled Data

- unlabeled data is usually used to learn a latent representation,
- and then the target regressed on the latent representation z, not the original representation x.



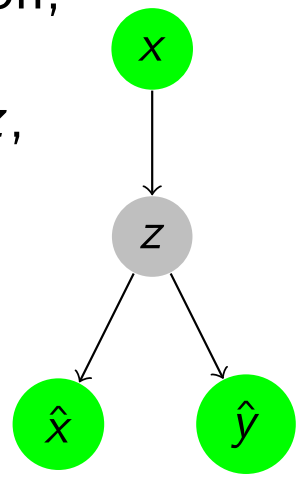
Semi-supervised Learning / Using the Unlabeled Data

- unlabeled data is usually used to learn a latent representation,
- and then the target regressed on the latent representation z, not the original representation x.
- sequential training:
  - first train $\hat{x}$ (and z), then fix z and train $\hat{y}$.
  - advantage: simple.
  - disadvantage: latent representation is not informed by the task $\hat{y}$.



Semi-supervised Learning / Using the Unlabeled Data

- unlabeled data is usually used to learn a latent representation,
- and then the target regressed on the latent representation z, not the original representation x.



- sequential training:
  - first train $\hat{x}$ (and z), then fix z and train $\hat{y}$.
  - advantage: simple.
  - disadvantage: latent representation is not informed by the task $\hat{y}$.
- concurrent training:
  - alternate between training $\hat{x}$ and $\hat{y}$.
  - joint loss:

$$\ell(\hat{y}, \hat{x}; \mathcal{D}^{\text{train}}, \mathcal{D}^{\text{unlab}}) = \ell_Y(\hat{y}; \mathcal{D}^{\text{train}}) + \alpha \ell_X(\hat{x}; \mathcal{D}^{\text{train}}|_x \cup \mathcal{D}^{\text{unlab}})$$

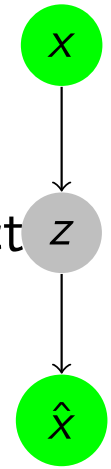
  - auxiliary loss weight α and latent dimensionality K are hyperparameters (as well as the choice of the reconstruction loss ℓ_X)

Outline

Autoencoders / Single Layer

- PCA / linear autoencoder:

$$\hat{x}(x) := VWx \rightsquigarrow W^2W^1x \rightsquigarrow W^2(W^1x + b^1) + b^2$$



- autoencoder: chose encoder and decoder to be a fully connect neural network with L layers each.
- most simple case: a single layer $L = 1$:

$$\hat{x}(x) := \text{relu}(W^2 \text{relu}(W^1x + b^1) + b^2)$$

Autoencoders / Multiple Layers

- general case: multiple layers L each:
- encoder block $z^{1,1:L_1}$:

$$z^{1,\ell} := \text{relu}(W^{1,\ell} z^{1,\ell-1} + b^{1,\ell})$$

$$z^{1,0} := x, \quad M_{1,0} := M, \quad h := z^{1,L}, \quad M_{1,L} := K,$$

$$W^{1,\ell} \in \mathbb{R}^{M_{1,\ell} \times M_{1,\ell-1}}, \quad b^{1,\ell} \in \mathbb{R}^{M_{1,\ell}}$$

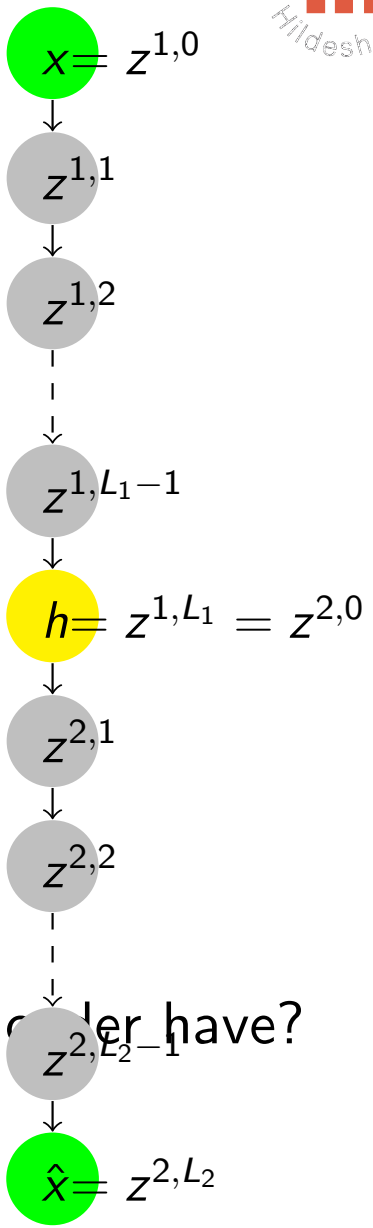
- decoder block $z^{2,1:L_2}$:

$$z^{2,\ell} := \text{relu}(W^{2,\ell} z^{2,\ell-1} + b^{2,\ell})$$

$$z^{2,0} := h, \quad M_{1,0} := K, \quad \hat{x} := z^{2,L}, \quad M_{2,L} := M,$$

$$W^{2,\ell} \in \mathbb{R}^{M_{2,\ell} \times M_{2,\ell-1}}, \quad b^{2,\ell} \in \mathbb{R}^{M_{2,\ell}}$$

- Q: What type of architecture does such a deep autoencoder have?



Autoencoders / Multiple Layers

- general case: multiple layers L each:
- encoder block $z^{1,1:L_1}$:

$$z^{1,\ell} := \text{relu}(W^{1,\ell} z^{1,\ell-1} + b^{1,\ell})$$

$$z^{1,0} := x, \quad M_{1,0} := M, \quad h := z^{1,L}, \quad M_{1,L} := K,$$

$$W^{1,\ell} \in \mathbb{R}^{M_{1,\ell} \times M_{1,\ell-1}}, \quad b^{1,\ell} \in \mathbb{R}^{M_{1,\ell}}$$

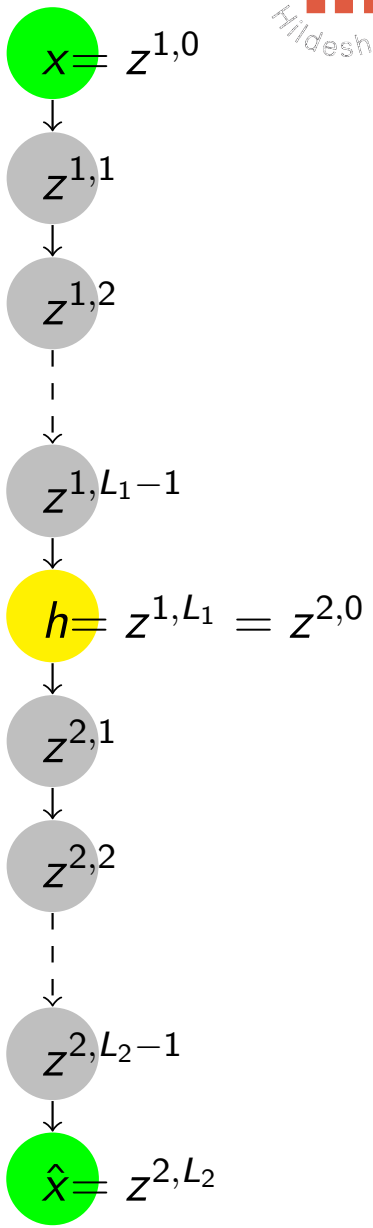
- decoder block $z^{2,1:L_2}$:

$$z^{2,\ell} := \text{relu}(W^{2,\ell} z^{2,\ell-1} + b^{2,\ell})$$

$$z^{2,0} := h, \quad M_{1,0} := K, \quad \hat{x} := z^{2,L}, \quad M_{2,L} := M,$$

$$W^{2,\ell} \in \mathbb{R}^{M_{2,\ell} \times M_{2,\ell-1}}, \quad b^{2,\ell} \in \mathbb{R}^{M_{2,\ell}}$$

- an autoencoder is a vanilla fully connected feedforward neural network

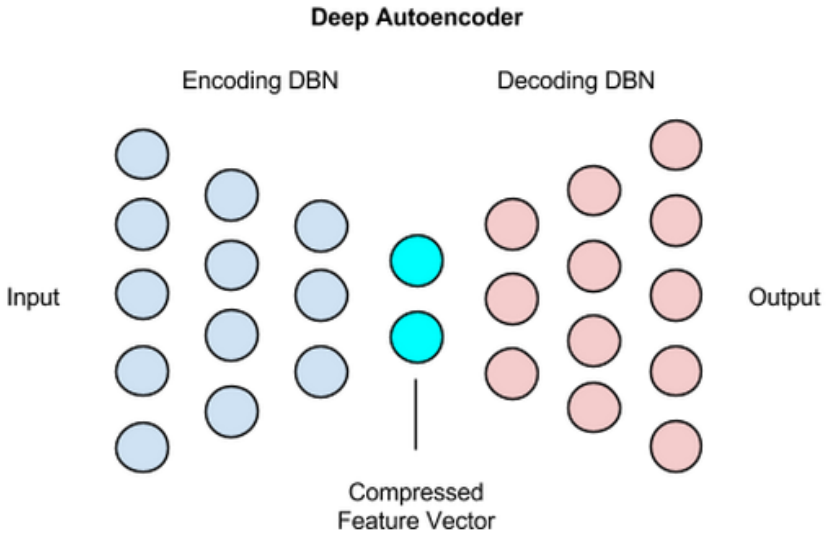


Autoencoders

- usually, depth and layer sizes of encoder and decoder blocks are chosen antisymmetric:

$$L_1 = L_2, \quad M_{2,\ell} = M_{1,L_1-\ell}$$

- create the information bottleneck gradually, layer by layer.



[source: licdn.com]

Learning Autoencoders

- an autoencoder is a vanilla fully connected feedforward neural network.
- thus an autoencoder is learnt via vanilla backpropagation.
- from unlabeled data $\mathcal{D} = (x_1, x_2, \ldots, x_N) \in (R^M)^*$
  to labeled training data for autoencoders:

$$\mathcal{D}^{\text{train}} := ((x_1, x_1), (x_2, x_2), \ldots, (x_N, x_N)) \in (\mathbb{R}^M \times \mathbb{R}^M)^*$$

$$\ell(x, \hat{x}; \mathcal{D}^{\text{train}}) := \frac{1}{N} \sum_{n=1}^{N} ||x_n - \hat{x}(x_n)||_2^2 = \frac{1}{N} \sum_{n=1}^{N} \sum_{m=1}^{M} (x_{n,m} - \hat{x}_m(x_n))^2$$

Regularizing Autoencoders

- one-layer autoencoders are often just structurally regularize: $K \ll M$.
- deeper autoencoders and more generally, a more fine-grainded regularization is useful.
- use standard regularizations such as L2, drop-out

$$f(\theta) := \ell(\hat{x}; \theta) + \lambda ||\theta||_2^2$$

  - also L1 regularization is possible:

$$f(\theta) := \ell(\hat{x}; \theta) + \lambda ||\theta||_1$$

Enforcing Properties of the Latent Representations

- **sparse autoencoders**:

$$f(\theta; \mathcal{D}^{\text{train}}) := \ell(\hat{x}; \theta, \mathcal{D}^{\text{train}}) + \lambda \frac{1}{N} \sum_{n=1}^{N} ||h(x_n; \theta)||_1$$

  - encourages sparse latent representations $h(x_n)$.
  - thus also has a regularizing effect for the encoder.

Denoising Autoencoders

- goal: make the autoencoder more robust to small variations in the input.
- corrupt the input with noise.
- still require the autoencoder to decode to the clean, original instance:

$$\ell(\hat{x}; \mathcal{D}^{\text{train}}) := \mathbb{E}_{\epsilon \sim p_{\text{noise}}} \ell(x, \hat{x}(z(x + \epsilon)))$$

  - e.g., $p_{\text{noise}} := \mathcal{N}_M(0, \sigma^2)$ with a small noise variance σ^2.
  - during training, draw a fresh corruption ϵ and add to instance x_n, before it is input into the network. So for each batch:

$$\mathcal{D}^{\text{train}}_{\text{noise}} := ((x_n + \epsilon_n, x_n) \mid n = 1 : N, \epsilon_n \sim p_{\text{noise}})$$

  - basically a data augmentation technique.

Contractive Autoencoders

- goal: make the autoencoder more robust to small variations in the input.
- penalize large gradients of the latent representaitons h w.r.t. the inputs x:

$$f(\theta; \mathcal{D}^{\text{train}}) := \ell(\hat{x}; \theta, \mathcal{D}^{\text{train}}) + \lambda \frac{1}{N} \sum_{n=1}^{N} ||\frac{\partial h(x_n; \theta)}{\partial x}||_F^2$$

- e.g., for a single layer autoencoder:

$$||\frac{\partial h(x_n; \theta)}{\partial x}||_F^2 = \sum_{k=1}^{K} \sum_{m=1}^{M} (\frac{\partial h_k(x_n)}{\partial x_m})^2 = \sum_{k=1}^{K} \sum_{m=1}^{M} (a'(W_{k.}^T x + b_k) W_{k,m})^2$$

$$\underset{\text{i.g.}}{\neq} ||W||_F^2$$

Note: Remember the Frobenius matrix norm $||A||_F := \sum_{n=1}^{N} \sum_{m=1}^{M} (A_{n,m})^2$.

Outline

Layers Parametrizing Distributions

- univariate normal ($K = 1$):

$$M_{\ell-1} := 2, \quad \mu := z_1^{\ell-1}, \quad \sigma := e^{z_2^{\ell-1}}$$
$$p(z^\ell \mid z^{\ell-1}) := \mathcal{N}(z^\ell \mid \mu, \sigma^2)$$

- multivariate normal (K), diagonal covariance matrix:

$$M_{\ell-1} := 2K, \quad \mu := z_{1:K}^{\ell-1}, \quad \Sigma := \mathrm{diag}(e^{2z_{K+1:2K}^{\ell-1}})$$
$$p(z^\ell \mid z^{\ell-1}) := \mathcal{N}_K(z^\ell \mid \mu, \Sigma)$$

- multivariate normal (K), full covariance matrix:

$$M_{\ell-1} := \frac{K(K+1)}{2}, \quad \mu := z_{1:K}^{\ell-1}, \quad A := \mathrm{reshape}(z_{K+1:\frac{K(K+1)}{2}}^{\ell-1}), \quad \Sigma := A^T A$$
$$p(z^\ell \mid z^{\ell-1}) := \mathcal{N}_K(z^\ell \mid \mu, \Sigma)$$

Note: Here, reshape reshapes the $\frac{K(K-1)}{2}$ entries into a lower triangular $K \times K$ matrix A with zeros on the diagonal.

Layers Parametrizing Distributions (2/2)

- bernoulli:

$$M_{\ell-1} := 1, \quad p_1 := \text{logistic}(z_1^{\ell-1})$$
$$p(z^\ell \mid z^{\ell-1}) := \text{Ber}(z^\ell \mid p_1) = z^\ell p_1 + (1 - z^\ell)(1 - p_1)$$

- multinoulli (K):

$$M_{\ell-1} := K, \quad p := \text{softmax}(z^{\ell-1})$$
$$p(z^\ell \mid z^{\ell-1}) := \text{Mult}(z^\ell \mid p) = \prod_{k=1}^{K} p_k^{\mathbb{I}(z^\ell=k)}$$

Layers Parametrizing Distributions / Value Computation

- value computation / forwards:
  - sample $z^\ell \sim p(z^\ell \mid z^{\ell-1})$
  - to estimate output probabilities, take S samples:

$$p(x \mid z) = \frac{1}{S} \sum_{s=1}^{S} p(x \mid z^{\ell,(s)}), \quad z^{\ell,(s)} \sim p(z^\ell \mid z^{\ell-1})$$

- Q: but how to compute gradients $\frac{\partial z^\ell}{\partial z^{\ell-1}}$ required for backpropagation?

Layers Parametrizing Distributions / Gradient Computation

- gradient computation / backwards
- **reparameterization trick**:
  - reparametrize z^ℓ as a differentiable function in
    - the previous layer and
    - random noise ϵ (independent from the previous layer):

$$z^\ell \mid z^{\ell-1} = g(z^{\ell-1}, \epsilon), \quad \epsilon \sim p_{\text{noise}}$$

  - e.g., univariate normal:

$$p(z^\ell \mid z^{\ell-1}) = \mathcal{N}(z^\ell \mid \mu, \sigma^2) \quad \rightsquigarrow \quad z^\ell = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(\epsilon \mid 0, 1)$$

  - now $\frac{\partial z^\ell}{\partial z^{\ell-1}}$ is straight-forward to compute.

Layers Parametrizing Distributions / Gradient Computation

- reparametrization trick for multivariate normal with diagonal covariance:

$$
\begin{aligned}
p(z^\ell \mid z^{\ell-1}) &= \mathcal{N}_K(z^\ell \mid \mu, \mathrm{diag}(\sigma_1^2 \ldots, \sigma_K^2)) \\
\rightsquigarrow \quad z^\ell &= \mu + (\sigma_1, \ldots, \sigma_K)^T \odot \epsilon, \quad \epsilon \sim \mathcal{N}_K(\epsilon \mid 0, 1) \\
&= \mu + \mathrm{diag}(\sigma_1, \ldots, \sigma_K)\epsilon
\end{aligned}
$$

- reparametrization trick for multivariate normal with full covariance:

$$
\begin{aligned}
p(z^\ell \mid z^{\ell-1}) &= \mathcal{N}_K(z^\ell \mid \mu, \Sigma), \quad \text{with } \Sigma = A^T A \\
\rightsquigarrow \quad z^\ell &= \mu + A\epsilon, \quad \epsilon \sim \mathcal{N}_K(\epsilon \mid 0, 1)
\end{aligned}
$$

Layers Parametrizing Distributions / Gradient Computation

- neural networks with layers parametrizing distributions can be learnt with vanilla backpropagation
  - after applying the reparametrization trick
  - random noise ϵ is viewed as additional input
- for autoencoders with point reconstructions $\hat{x}(z)$:

$$\min f(\theta; \mathcal{D}) := \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}_{z \sim p(z|x_n)} \ell(x_n, \hat{x}(z))$$

- for autoencoders with distributions $p(\hat{x} \mid z)$ of reconstructions:

$$\min f(\theta; \mathcal{D}) := -\frac{1}{N} \sum_{n=1}^{N} \mathbb{E}_{z \sim p(z|x_n)} \log p(\hat{x} = x_n \mid z)$$

Outline

Variational Lower Bound

- variational autoencoders distinguish between
  - the true latent posterior $p(z \mid x)$ and
  - the estimated latent posterior $p(\hat{z} \mid x)$
    — usually denoted $q(z \mid x)$ in the literature.
  - maximize log likelihood:

$$
\begin{aligned}
&\log\, p(\hat{x} = x) \\
&\quad = \mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \log p(\hat{x} = x) \\
&\quad = \mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \log \frac{p(\hat{x} = x, z = \hat{z})}{p(z = \hat{z} \mid x)} \\
&\quad = \mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \log \frac{p(\hat{x} = x, z = \hat{z})}{p(\hat{z} \mid x)} \frac{p(\hat{z} \mid x)}{p(z = \hat{z} \mid x)} \\
&\quad = \mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \log p(\hat{x} = x, z = \hat{z}) - \mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \log p(\hat{z} \mid x) \\
&\qquad\qquad + \mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \frac{\log p(\hat{z} \mid x)}{\log p(z = \hat{z} \mid x)}
\end{aligned}
$$

Variational Lower Bound

$$\log p(\hat{x} = x) = \mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \log p(\hat{x} = x, z = \hat{z}) \overbrace{-\mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \log p(\hat{z} \mid x)}^{=H(p(\hat{z}|x))}$$

$$+ \underbrace{\mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \frac{\log p(\hat{z} \mid x)}{\log p(z = \hat{z} \mid x)}}_{=\mathrm{KL}(p(\hat{z}|x), p(z|x))}$$

$$= \mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \log p(\hat{x} = x, z = \hat{z}) + H(p(\hat{z} \mid x)) + \mathrm{KL}(p(\hat{z} \mid x), p(z \mid x))$$

- Entropy $H(q)$
- Kullback-Leibler divergence $\mathrm{KL}(q, p) \geq 0$
  - cannot be computed as $p(z = \hat{z} \mid x)$ is not accessible, only conceptual.
  - drop it and use the remaining terms as a lower bound.

Variational Lower Bound

- **evidence lower bound (ELBO) / variational lower bound**:

$$\begin{aligned}\text{ELBO} &:= \log p(\hat{x} = x) - \text{KL}(p(\hat{z} \mid x), p(z \mid x)) \\ &= \mathbb{E}_{\hat{z} \sim p(\hat{z}|x)} \log p(\hat{x} = x, z = \hat{z}) + H(p(\hat{z} \mid x))\end{aligned}$$

- conceptually:
  - maximize the likelihood of the data and
  - minimize the KL divergence, i.e.,

    make estimated posterior latent distribution $p(\hat{z} \mid x)$ and true posterior latent distribution $p(z \mid x)$ similar.
- technically:
  - maximize the joint likelihood of the data and the estimated latent representations, and
  - maximize the entropy of the estimated latent representations.

Variational Lower Bound

- entropy for univariate normal:

$$H(\mathcal{N}(\mu, \sigma^2)) \propto \log \sigma$$

- entropy for multivariate normal with diagonal covariance matrix:

$$H(\mathcal{N}_K(\mu, (\sigma_1^2, \ldots, \sigma_K^2)) \propto \sum_{k=1}^{K} \log \sigma_k$$

- entropy for multivariate normal with full covariance matrix:

$$H(\mathcal{N}(\mu, \Sigma)) \propto \ldots$$

Variational Autoencoders

- Variational Autoencoders are autoencoders with
  1. a layer parametrizing a normal distribution as latent representation layer h,
     - usually with a diagonal covariance matrix (factorized Gaussian posteriors).

$$M_{1,\ell-1} := 2K, \quad \mu := z^{\ell-1}_{1,1:K}, \quad \Sigma := \mathrm{diag}(e^{2z^{\ell-1}_{K+1:2K}})$$
$$p(z^\ell \mid z^{\ell-1}) := \mathcal{N}_K(z^\ell \mid \mu, \Sigma)$$

  2. using the variational lower bound ELBO as loss
     - i.e., for a diagonal covariance matrix and scalar normal output $\mathcal{N}(\hat{x} \mid \mu_{\text{out}}(x_n, \hat{z}), \sigma^2_{\text{out}}(x_n, \hat{z}))$, minimize:

$$f(\theta) = -\,\text{ELBO}$$
$$= -\frac{1}{N}\Big(\sum_{n=1}^{N} \mathbb{E}_{\hat{z}\sim p(\hat{z}|x_n)} \log p(\hat{x} = x_n, z = \hat{z}) + \sum_{k=1}^{K} \log(\Sigma_{k,k}(x_n))\Big)$$
$$= \frac{1}{N}\Big(\sum_{n=1}^{N} \mathbb{E}_{\hat{z}\sim p(\hat{z}|x_n)} \log(\sigma_{\text{out}}(x_n, \hat{z})) + \Big(\frac{x_n - \mu_{\text{out}}(x_n, \hat{z})}{\sigma_{\text{out}}(x_n, \hat{z})}\Big)^2 - 2\sum_{k=1}^{K} z^{\ell-1}_{K+1:2K}(x$$

Summary

- Dimensionality reduction can be accomplished via **reconstructing the features** of each instance from a lower dimensional / lower complexity latent representation (**autoencoding**).
- **Principal Components Analysis** (**PCA**) is a linear autoencoder.
- Adding a non-linear activation function to the latent layer yields a **single layer autoencoder**.
- More generally, any feed-forward neural network can be interpreted as **deep autoencoder**,
  - picking any of its layers as latent representation h,
  - the layers up to h are interpreted as **encoder**,
  - the layers after h are interpreted as **decoder**.
  - but usually, layer sizes are chosen anti-symmetric and the middle layer is chosen as latent representation h.

Summary (2/2)

- Autoencoders are vanilla feed-forward neural networks, esp.
  - they can be trained via vanilla back-propagation.
  - they should be regularized.
    - standard: L2, drop-out
    - special: **sparse autoencoders** with an L1 regularization on the latent representation h
- **Layers representing distributions**
  - can be modeled by parametrizing distributions by values from the previous layer.
  - can be used forwards simply by sampling from the distribution.
  - can compute gradients by the **reparametrization trick**.
- **Variational Autoencoders**
  - represent the latent features by a distribution (not just a point estimate)
    - usually a multivariate normal with diagonal covariance matrix,
  - use a **variational lower bound ELBO** as loss.

Further Readings

- Goodfellow et al. 2016, ch. 13 & 14.
- variational autoencoders:
  - Kingma and Welling [2019]
  - Goodfellow et al. 2016, ch. 20.
- autoencoders are not covered explicitely by Zhang et al. 2020.

Acknowledgement: An earlier version of the slides for this lecture have been written by my former postdoc Dr Josif Grabocka.

References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The Mit Press, Cambridge, Massachusetts, November 2016. ISBN 978-0-262-03561-3.

Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander Smola. *Dive into Deep Learning*. https://d2l.ai/, 2020.