CS-E5875 High-Throughput Bioinformatics
Machine learning for scRNA-seq analysis

Harri Lähdesmäki

Department of Computer Science
Aalto University

December 1, 2023

Contents

Generalized linear model for binary-valued data

- Recall again the generalized linear modeling (GLM) framework
- Consider data $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i = (x_{i1}, \ldots, x_{ik})^T$ denotes the explanatory variables and the response variable y_i can have only two possible value: $\{0, 1\}$
- Binary data can be modeled using the Bernoulli probability density function:

$$\mathrm{Ber}(y \mid p) = p^y (1 - p)^{1-y},$$

where p is the probability of success, or the mean (parameter) as $\mathbb{E}(Y) = p$

Generalized linear model for binary-valued data

- Recall again the generalized linear modeling (GLM) framework
- Consider data $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$, where $\mathbf{x}_i = (x_{i1}, \ldots, x_{ik})^T$ denotes the explanatory variables and the response variable y_i can have only two possible value: $\{0, 1\}$
- Binary data can be modeled using the Bernoulli probability density function:

$$\mathrm{Ber}(y \mid p) = p^y (1-p)^{1-y},$$

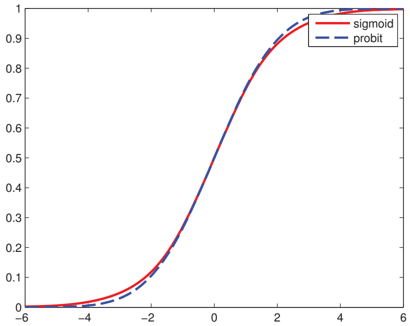
  where p is the probability of success, or the mean (parameter) as $\mathbb{E}(Y) = p$
- In GLM framework we want to model p using a linear model via a link function
- For binary data the following link function is useful

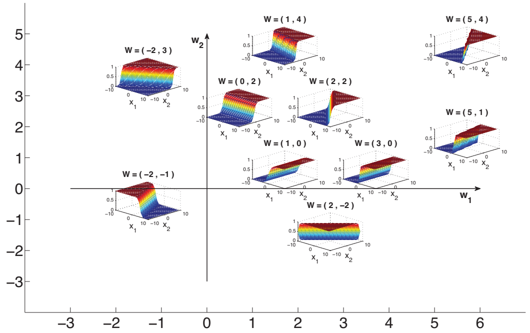
$$\log\left(\frac{p_i}{1-p_i}\right) = \mathbf{x}_i^T \boldsymbol{\beta} + \beta_0 \quad \Longleftrightarrow \quad p_i = \frac{1}{1+\exp(-(\mathbf{x}_i^T \boldsymbol{\beta} + \beta_0))} = \mathrm{sigm}(\mathbf{x}_i^T \boldsymbol{\beta} + \beta_0),$$

  where $\mathrm{sigm} : \mathbb{R} \to [0, 1]$ is the sigmoidal function that maps the real line to the interval between 0 and 1
- Therefore, our model is $\mathrm{Ber}(y_i \mid \mathrm{sigm}(\mathbf{x}_i^T \boldsymbol{\beta} + \beta_0))$

Logistic regression

- Machine learning terminology:
  - The sigmoidal function is called the activation function and denoted here as $\phi(\cdot)$
  - The GLM model for binary data is called the logistic regression model or linear classifier and is denoted as $y = f(\mathbf{x}) = \phi(\mathbf{x}^T\boldsymbol{\beta} + \beta_0) = \phi(\boldsymbol{\beta}^T\mathbf{x} + \beta_0)$
- Illustrations of the sigmoidal activation function (left) and linear classifiers for two covariates (right)





Figures from (Murphy, 2012)

Neural networks: Perceptron

- Some machine learning / neural network models are inspired by neuroscience and can be seen as models which try to mimic information processing in brain
- The first neural network model by Rosenblatt was called perceptron
- Perceptron is essentially the logistic regression model where the activation function is the step function
$$y = f(\mathbf{x}) = \mathrm{sign}(\boldsymbol{\beta}^T \mathbf{x} + \beta_0),$$
where $\mathrm{sign}(x) = 0$ if $x < 0$, and $\mathrm{sign}(x) = 1$ if $x \geq 0$
- Term perceptron is nowadays used to denote the logistic regression model with an activation function that is typically something else than the step function
- Linear classifier and perceptron are limited in that they can only solve linear classification problems (where two classes are linearly separable)

Multilayer perceptron

- Multilayer perceptron (MLP) is the most basic type of deep neural network model
- MLP combines several linear classifiers (perceptrons) such that outputs of the perceptrons in the previous layer are used as the inputs to the linear classifier in the next layer
- Each node in the network implements the function

$$y = f(\mathbf{x}) = \phi(\mathbf{w}^T\mathbf{x} + b),$$

where $\mathbf{w}$ and b are the linear model weights that are different for each node

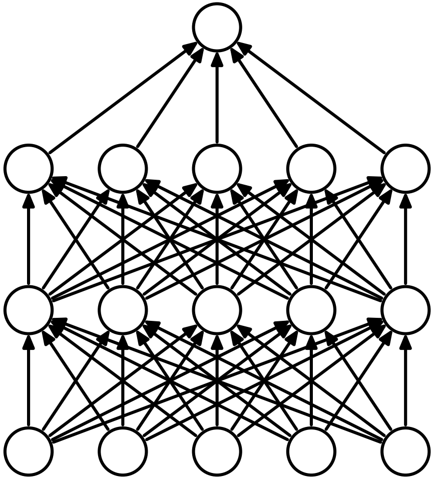


Figure from (Murphy, 2020)

Multilayer perceptron

- The nodes at the bottom correspond to the input $\mathbf{x}$
- $\mathbf{h}_1$ denotes the outputs of the perceptrons in the first layer ($\mathbf{W}_1 = (\mathbf{w}_{11}, \dots, \mathbf{w}_{1m})$)

$$\begin{aligned}\mathbf{h}_1 &= (\phi(\mathbf{w}_{11}^T\mathbf{x} + b_{11}), \dots, \phi(\mathbf{w}_{1m}^T\mathbf{x} + b_{1m}))^T \\ &= \phi(\mathbf{W}_1^T\mathbf{x} + \mathbf{b}_1),\end{aligned}$$

- $\mathbf{h}_2$ denotes the outputs of the perceptrons in the 2nd layer ($\mathbf{W}_2 = (\mathbf{w}_{21}, \dots, \mathbf{w}_{2m})$)

$$\begin{aligned}\mathbf{h}_2 &= (\phi(\mathbf{w}_{21}^T\mathbf{h}_1 + b_{21}), \dots, \phi(\mathbf{w}_{2m}^T\mathbf{h}_1 + b_{2m}))^T \\ &= \phi(\mathbf{W}_2^T\mathbf{h}_1 + \mathbf{b}_2)\end{aligned}$$

- Output is $y = \psi(\mathbf{w}_3^T\mathbf{h}_2 + b_3)$

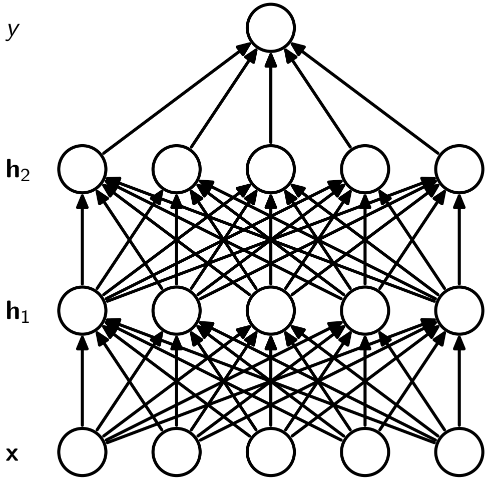


Figure from (Murphy, 2020)

Multilayer perceptron

- The MLP model can be written more compactly as

  $$y = f_3(\mathbf{f}_2(\mathbf{f}_1(\mathbf{x}; \boldsymbol{\theta}_1); \boldsymbol{\theta}_2); \boldsymbol{\theta}_3),$$

  where $\mathbf{f}_1(\cdot; \boldsymbol{\theta}_1)$, $\mathbf{f}_2(\cdot; \boldsymbol{\theta}_2)$ and $f_3(\cdot; \boldsymbol{\theta}_3)$ are the perceptrons from the previous page (that have outputs $\mathbf{h}_1$, $\mathbf{h}_2$, y) and $\boldsymbol{\theta}_i = (\mathbf{W}_i, \mathbf{b}_i)$, $\boldsymbol{\theta}_i = (\mathbf{W}_i, \mathbf{b}_i)$, and $\boldsymbol{\theta}_3 = (\mathbf{w}_3, b_3)$
- Alternatively write

  $$y = f(\mathbf{x}; \boldsymbol{\theta}),$$

  where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3)$

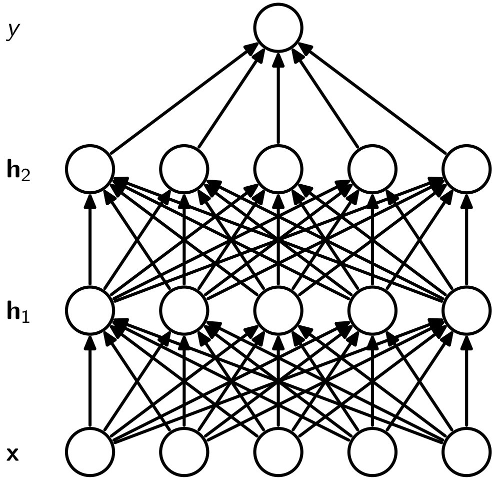


Figure from (Murphy, 2020)

Multilayer perceptron:

- Although our example MLP has two hidden layers, in general MLPs can have any number of layers
- Each layer can have an arbitrary number of nodes (=width)
- MLPs can use different types of activation functions

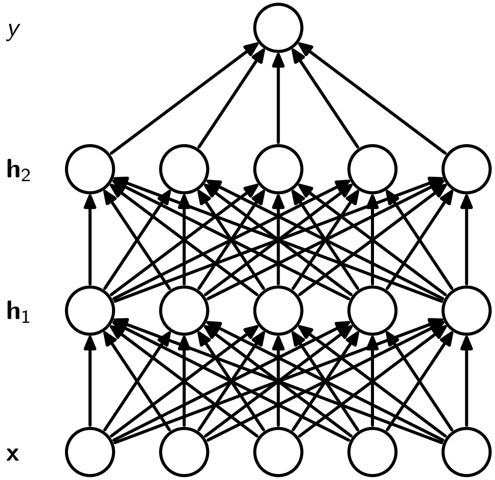


Figure from (Murphy, 2020)

Multilayer perceptron: likelihood and inference

- Binary classification: choose the activation function for the last layer to be e.g. the sigmoidal function $\psi(\cdot) = \mathrm{sigm}(\cdot)$ and use the Bernoulli likelihood for the data D

$$\mathcal{L}(\boldsymbol{\theta}) = p(D \mid \boldsymbol{\theta}) = \prod_{i=1}^{n} \mathrm{Ber}(y_i \mid f(\mathbf{x}_i; \boldsymbol{\theta}))$$

Multilayer perceptron: likelihood and inference

- Binary classification: choose the activation function for the last layer to be e.g. the sigmoidal function $\psi(\cdot) = \mathrm{sigm}(\cdot)$ and use the Bernoulli likelihood for the data D

$$\mathcal{L}(\boldsymbol{\theta}) = p(D \mid \boldsymbol{\theta}) = \prod_{i=1}^{n} \mathrm{Ber}(y_i \mid f(\mathbf{x}_i; \boldsymbol{\theta}))$$

- Regression assuming additive Gaussian noise: the activation function for the last layer can be the identity function and use the Gaussian likelihood

$$\mathcal{L}(\boldsymbol{\theta}) = p(D \mid \boldsymbol{\theta}) = \prod_{i=1}^{n} \mathcal{N}(y_i \mid f(\mathbf{x}_i; \boldsymbol{\theta}), \sigma^2)$$

Multilayer perceptron: likelihood and inference

- Binary classification: choose the activation function for the last layer to be e.g. the sigmoidal function $\psi(\cdot) = \mathrm{sigm}(\cdot)$ and use the Bernoulli likelihood for the data D

$$\mathcal{L}(\boldsymbol{\theta}) = p(D \mid \boldsymbol{\theta}) = \prod_{i=1}^{n} \mathrm{Ber}(y_i \mid f(\mathbf{x}_i; \boldsymbol{\theta}))$$

- Regression assuming additive Gaussian noise: the activation function for the last layer can be the identity function and use the Gaussian likelihood

$$\mathcal{L}(\boldsymbol{\theta}) = p(D \mid \boldsymbol{\theta}) = \prod_{i=1}^{n} \mathcal{N}(y_i \mid f(\mathbf{x}_i; \boldsymbol{\theta}), \sigma^2)$$

- The large number of parameters of the model can be chosen by maximizing the likelihood
- No closed form solution but the parameters can be optimized iteratively using numerical optimization methods

$$\boldsymbol{\theta}^{(s+1)} := \boldsymbol{\theta}^{(s)} + \Delta \frac{\partial \log \mathcal{L}}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}^{(s)}}$$

Multilayer perceptron: illustration

- Deep MLP can learn complex functions

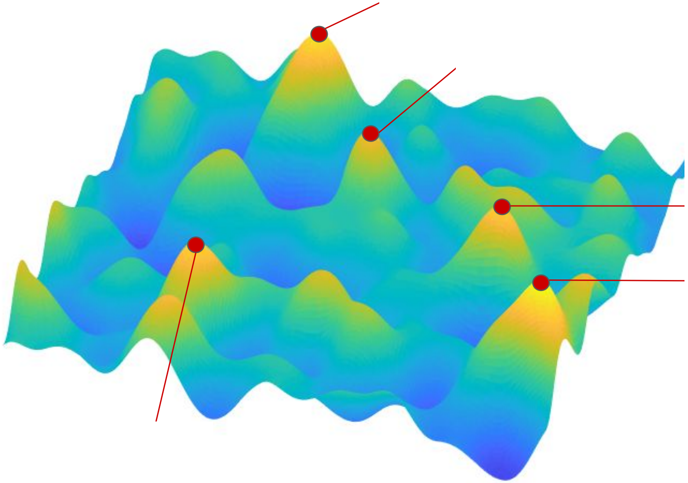


Figure by Cagatay Yildiz

Contents

Cell type identification with neural networks

- Computational cell type identification is an important step in scRNA-seq analysis
- Several cell type annotation methods rely on unsupervised clustering
- We we will look at a supervised cell type annotation method that uses deep learning method which is trained on a data set of labelled single cells

Automated cell type identification using neural networks (ACTINN)

- ACTINN (Ma and Pellegrini, 2019) is one of the many recently proposed supervised methods for cell type identification that use deep learning methods
- Labelled single-cell gene expression profiles $D = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$ are collected from different databases
- The number of cell types is denoted by k, $y_i \in \{1, \ldots, k\}$
- The method uses only those genes that appear in all the databases
- Genes with the highest 1% and lowest 1% mean expression are ignored
- Genes with the highest 1% and lowest 1% standard deviation are ignored
- The normalized gene expression vector $\mathbf{x}_i \in \mathbb{R}^d$ for d genes in cell i is obtained as

$$\mathbf{x}_i = \log_2 \frac{10^4 \cdot (\tilde{\mathbf{x}}_i + \mathbf{1})}{N_i},$$

  where $\tilde{\mathbf{x}}_i$ denotes the raw gene expression counts for all d genes and N_i is the total gene expression (count) measured for cell i

Automated cell type identification using neural networks (ACTINN)

- ACTINN uses the MLP with three hidden layers that have widths 100, 50 and 25

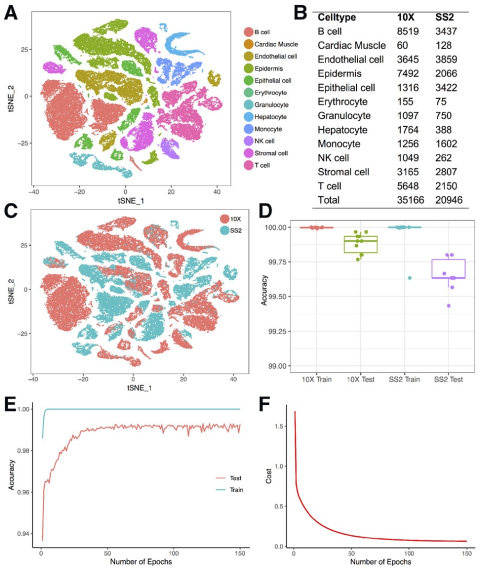
  $$y = f_4(\mathbf{f}_3(\mathbf{f}_2(\mathbf{f}_1(\mathbf{x}; \boldsymbol{\theta}_1); \boldsymbol{\theta}_2); \boldsymbol{\theta}_3); \boldsymbol{\theta}_4)$$

  with Relu activation function $\mathrm{Relu}(h) = \max(0, h)$ for the hidden layers and the softmax activation (or link function) for f_4

  $$\mathrm{softmax}(\mathbf{h}) = \left(\frac{\exp(\mathbf{h}(1))}{\sum_{j=1}^{k} \exp(\mathbf{h}(j))}, \cdots, \frac{\exp(\mathbf{h}(k))}{\sum_{j=1}^{k} \exp(\mathbf{h}(j))} \right)$$

  that maps the neural network outputs to k probabilities that sum up to one and thus allows modeling k cell types
- The model is trained using the multi-class extension of the Bernoulli likelihood (i.e., likelihood for categorical random variable) based loss function

Automated cell type identification using neural networks (ACTINN)



Celltype	10X	SS2
B cell	8519	3437
Cardiac Muscle	60	128
Endothelial cell	3645	3859
Epidermis	7492	2066
Epithelial cell	1316	3422
Erythrocyte	155	75
Granulocyte	1097	750
Hepatocyte	1764	388
Monocyte	1256	1602
NK cell	1049	262
Stromal cell	3165	2807
T cell	5648	2150
Total	35166	20946

Figure from (Ma and Pellegrini, 2019)

Comparison of cell type Identification methods

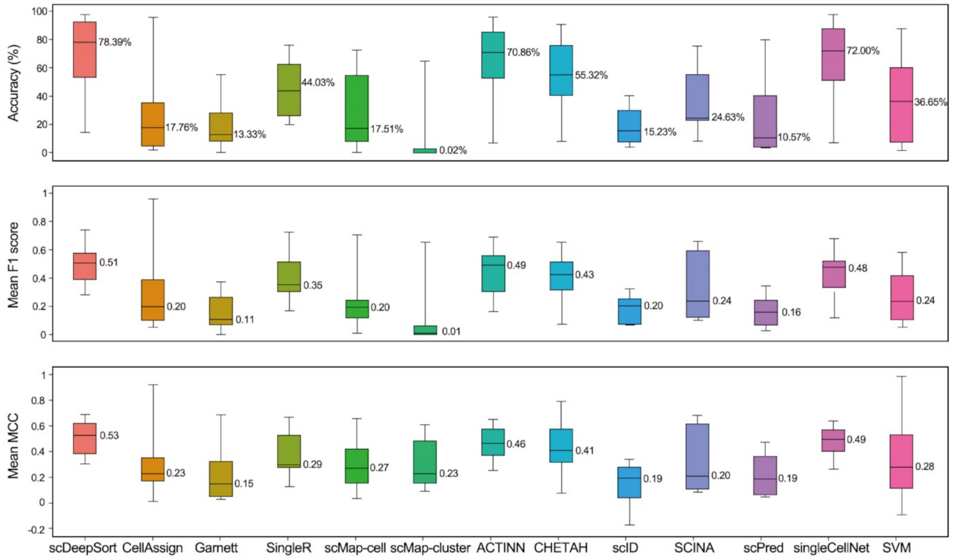


Figure from (Shao et al., 2021)

Contents

Dimension reduction

- In the previous lecture we discussed about PCA and how it can be used for dimension reduction and to analyze high-dimensional scRNA-seq data
- In this lecture we will look at some more advanced dimension reduction methods that can be presented as generative models

Probabilistic factor analysis

- Probabilistic factor analysis model can be described as the following generative model

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})$$
$$p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x} \mid \mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}),$$

  where $\mathbf{z} \in \mathbb{R}^L$ is a low-dimensional latent variable and $\mathbf{x} \in \mathbb{R}^D$ denotes observed data
- From data generation point of view, the model first samples a latent variable $\mathbf{z}$, and it then samples data vector $\mathbf{x}$ given a value for latent $\mathbf{z}$

Probabilistic factor analysis

- Probabilistic factor analysis model can be described as the following generative model

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})$$
$$p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x} \mid \mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi}),$$

  where $\mathbf{z} \in \mathbb{R}^L$ is a low-dimensional latent variable and $\mathbf{x} \in \mathbb{R}^D$ denotes observed data
- From data generation point of view, the model first samples a latent variable $\mathbf{z}$, and it then samples data vector $\mathbf{x}$ given a value for latent $\mathbf{z}$
- We can marginalize out the latent variable $\mathbf{z}$ from $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})$ to get

$$\begin{aligned} p(\mathbf{x}) &= \int p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})d\mathbf{z} \\ &= \int \mathcal{N}(\mathbf{x} \mid \mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi})\mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})d\mathbf{z} \\ &= \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi}) \end{aligned}$$

Probabilistic factor analysis illustration

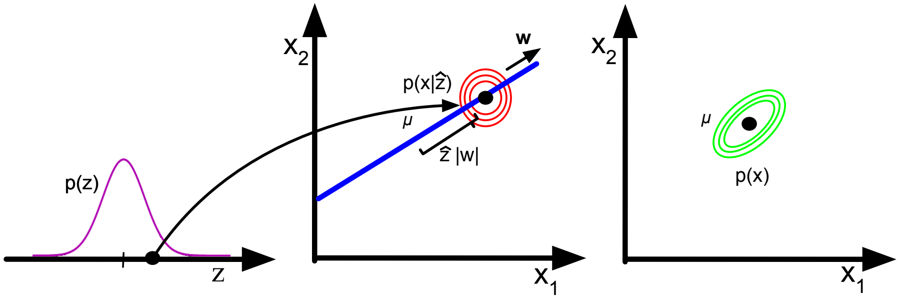


Figure 20.9: Illustration of the FA generative process, where we have $L = 1$ latent dimension generating $D = 2$ observed dimensions; we assume $\boldsymbol{\Psi} = \sigma^2\mathbf{I}$. The latent factor has value $z \in \mathbb{R}$, sampled from $p(z)$; this gets mapped to a 2d offset $\boldsymbol{\delta} = z\mathbf{w}$, where $\mathbf{w} \in \mathbb{R}^2$, which gets added to $\boldsymbol{\mu}$ to define a Gaussian $p(\mathbf{x}|z) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu} + \boldsymbol{\delta}, \sigma^2\mathbf{I})$. By integrating over z, we "slide" this circular Gaussian "spray can" along the principal component axis $\mathbf{w}$, which induces elliptical Gaussian contours in $\mathbf{x}$ space centered on $\boldsymbol{\mu}$. Adapted from Figure 12.9 of [Bis06].

Figure from (Murphy, 2020)

Probabilistic principle component analysis

- A special case of the FA model where columns of $\mathbf{W}$ are orthonormal, $\boldsymbol{\Psi} = \sigma^2\mathbf{I}$ and $\boldsymbol{\mu} = \mathbf{0}$ is called probabilistic principle component analysis (PPCA)

$$p(\mathbf{x}) = \int \mathcal{N}(\mathbf{x} \mid \mathbf{W}\mathbf{z}, \sigma^2\mathbf{I})\mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})d\mathbf{z} = \mathcal{N}(\mathbf{x} \mid \mathbf{0}, \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I})$$

Probabilistic principle component analysis

- A special case of the FA model where columns of $\mathbf{W}$ are orthonormal, $\mathbf{\Psi} = \sigma^2 \mathbf{I}$ and $\boldsymbol{\mu} = \mathbf{0}$ is called probabilistic principle component analysis (PPCA)

$$p(\mathbf{x}) = \int \mathcal{N}(\mathbf{x} \mid \mathbf{W}\mathbf{z}, \sigma^2 \mathbf{I}) \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I}) d\mathbf{z} = \mathcal{N}(\mathbf{x} \mid \mathbf{0}, \mathbf{W}\mathbf{W}^T + \sigma^2 \mathbf{I})$$

- Given an observed data set $D = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ we can estimate the model parameters $\mathbf{W}$ and σ^2
- Compute the sample covariance matrix $\mathbf{S} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$
- Rewrite $\mathbf{S}$ using the eigenvector-eigenvalue decomposition $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$
- Optimal parameters are $\mathbf{W} = \mathbf{U}_L(\mathbf{\Lambda}_L - \sigma^2 \mathbf{I})^{\frac{1}{2}}$ (upto arbitrary rotation) and $\sigma^2 = \frac{1}{D-L} \sum_{i=L+1}^{D} \lambda_i$

Probabilistic principle component analysis: posterior

- Given $\mathbf{W}$ and σ^2, we can use the PPCA and reduce the dimension of observed data $\mathbf{x}$
- The embedding of $\mathbf{x}$ can be shown to have normal distribution

$$p(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z} \mid \mathbf{M}^{-1}\mathbf{W}^T(\mathbf{x} - \boldsymbol{\mu}), \sigma^2\mathbf{M}^{-1}),$$

  where $\mathbf{M} = \mathbf{W}^T\mathbf{W} + \sigma^2\mathbf{I}$
- In the noise-free case of $\sigma^2 = 0$ the PPCA and PCA are directly comparable

Variational autoencoder

- The PPCA model is still a linear latent variable model
- We can extend the PPCA by defining the generative model to be nonlinear

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})$$
$$p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x} \mid f(\mathbf{z}; \theta), \sigma^2 \mathbf{I}),$$

  where $f(\cdot; \theta)$ is a nonlinear function which is typically parameterized by a deep neural network, such as the MLP
- For nonlinear models we can compute neither the marginal likelihood $p(\mathbf{x})$ nor the posterior of the latent representation $p(\mathbf{z} \mid \mathbf{x})$ analytically

Variational autoencoder

- The PPCA model is still a linear latent variable model
- We can extend the PPCA by defining the generative model to be nonlinear

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})$$
$$p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x} \mid f(\mathbf{z}; \theta), \sigma^2 \mathbf{I}),$$

  where $f(\cdot; \theta)$ is a nonlinear function which is typically parameterized by a deep neural network, such as the MLP
- For nonlinear models we can compute neither the marginal likelihood $p(\mathbf{x})$ nor the posterior of the latent representation $p(\mathbf{z} \mid \mathbf{x})$ analytically
- Another key idea of the variational autoencoder (VAE) model is to use so-called inference network $q(\mathbf{z} \mid \mathbf{x}, \phi)$ to approximate the intractable true posterior $p(\mathbf{z} \mid \mathbf{x})$
- If we assume that the variationally approximated posterior has normal distribution, then

$$q(\mathbf{z} \mid \mathbf{x}, \phi) = \mathcal{N}(\mathbf{z} \mid f_\mu(\mathbf{x}, \phi), f_{\sigma^2}(\mathbf{x}, \phi)),$$

  where $f_\mu(\cdot, \phi)$ and $f_{\sigma^2}(\cdot, \phi)$ are parametrized by deep neural network(s)

Variational autoencoder illustration

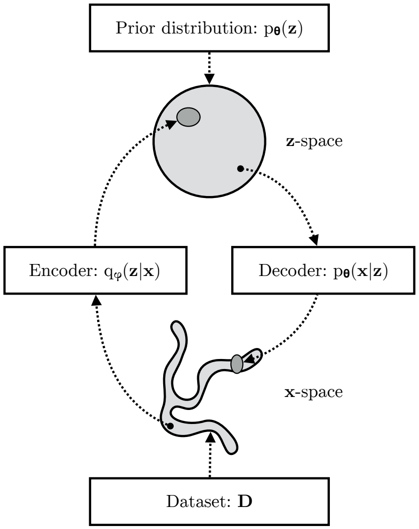

Prior distribution: $p_{\boldsymbol{\theta}}(\mathbf{z})$
$\mathbf{z}$-space
Encoder: $q_{\varphi}(\mathbf{z}|\mathbf{x})$
Decoder: $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
$\mathbf{x}$-space
Dataset: $\mathbf{D}$

Figure from (Kingma and Welling, 2019)

Variational autoencoder: training

- It can be shown that maximizing the marginal likelihood of the data $p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}$ corresponds to minimizing the Kullback-Leibler divergence from $q(\mathbf{z} \mid \mathbf{x}, \phi)$ to $p(\mathbf{z} \mid \mathbf{x})$

$$\mathrm{KL}(q(\mathbf{z} \mid \mathbf{x}, \phi) || p(\mathbf{z} \mid \mathbf{x})) = \int q(\mathbf{z} \mid \mathbf{x}, \phi) \log \frac{q(\mathbf{z} \mid \mathbf{x}, \phi)}{p(\mathbf{z} \mid \mathbf{x})} d\mathbf{z}$$

- This leads to so called evidence lower bound objective

$$\mathrm{ELBO} = \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x},\phi)} \log p(\mathbf{x} \mid \mathbf{z}, \theta)}_{\text{reconstruction term}} - \underbrace{\mathrm{KL}(q(\mathbf{z} \mid \mathbf{x}, \phi) || p(\mathbf{z}))}_{\text{regularization term}},$$

which can be maximized using a reparametrization trick and stochastic gradient methods

Contents

Deep generative models for single cell data

- scRNA-seq profiles contain both biological (mostly unknown) and technical (still poorly characterized) uncertainties
- Challenging to specify a well-motivated probabilistic model for the data
- VAE provides one principled modeling framework for complex scRNA-seq data

A variational autoencoder for scRNA-seq data

- Variational autoencoder architecture with deep neural networks

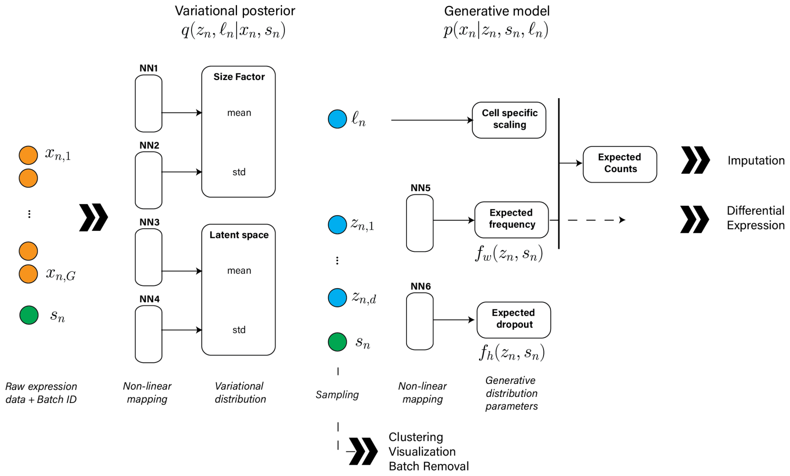


Figure from (Lopez et al, 2019)

A variational autoencoder for scRNA-seq data

- A probabilistic model for scRNA-seq data

$$
\begin{aligned}
z_n &\sim \text{Normal}(0, I) \\
\ell_n &\sim \text{LogNormal}(\ell_\mu, \ell_\sigma^2) \\
\rho_n &= f_w(z_n, s_n) \\
w_{ng} &\sim \text{Gamma}(\rho_n^g, \theta) \\
y_{ng} &\sim \text{Poisson}(\ell_n w_{ng}) \\
h_{ng} &\sim \text{Bernoulli}(f_h^g(z_n, s_n)) \\
x_{ng} &= \begin{cases} y_{ng} & \text{if } h_{ng} = 0, \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
$$

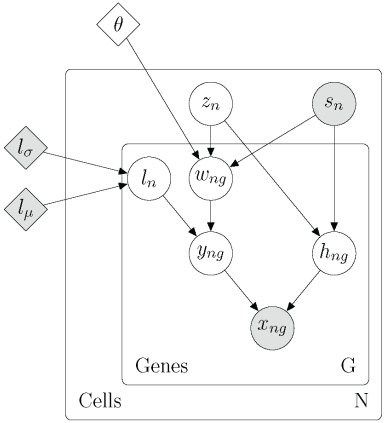


Figure from (Lopez et al, 2019)

A variational autoencoder for scRNA-seq data illustration

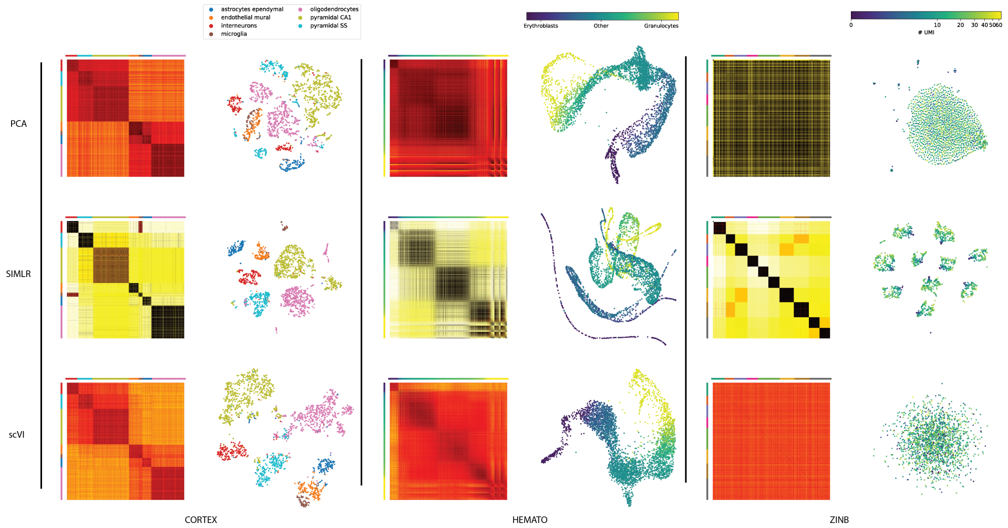


Figure from (Lopez et al, 2019)

References

- Kingma DP, Welling M, An Introduction to Variational Autoencoders, Foundations and Trends in Machine Learning, 2019
- R. Lopez, J. Regier, MB. Cole, M. Jordan, N. Yosef, Deep Generative Modeling for Single-cell Transcriptomics, *Nature Methods*, 2019
- Ma F, Pellegrini M, ACTINN: automated identification of cell types in single cell RNA sequencing, Bioinformatics, 36(2):533-538, 2019
- Murphy K, Machine learning: a probabilistic perspective, MIT Press, 2012
- Murphy K, Probabilistic machine learning: an introduction, MIT Press, 2019
- Shao X, et al., scDeepSort: a pre-trained cell-type annotation method for single-cell transcriptomics using deep learning with a weighted graph neural network, Nucleic Acids Research, https://doi.org/10.1093/nar/gkab775 , 2021