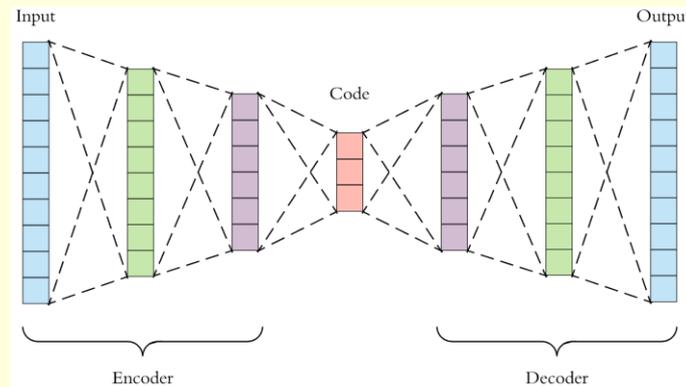


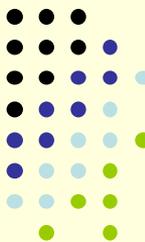
Deep Learning by Example on Biowulf

Class #3. Dimensionality reduction with Autoencoders

Gennady Denisov, PhD



Intro and goals



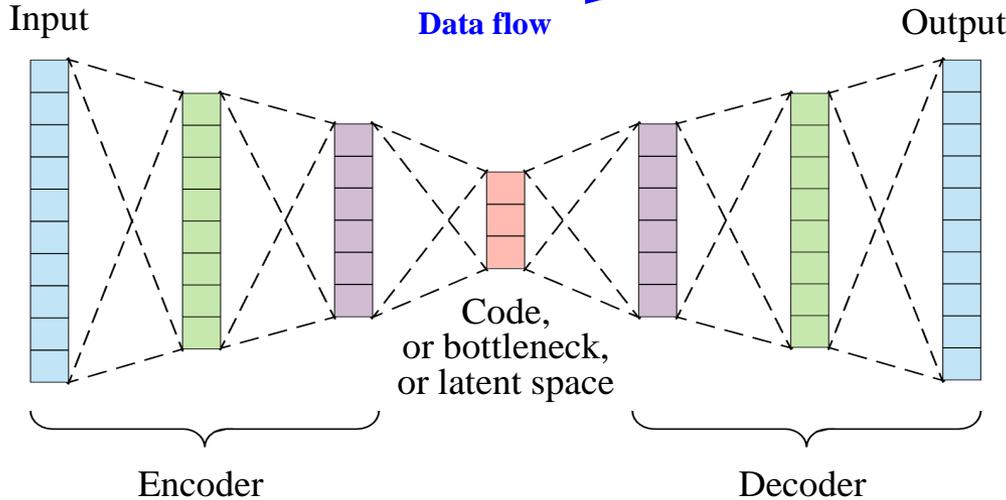
What is autoencoder?

Two basic requirements:

- 1) The sizes of the input and output tensors must be the same
- 2) At least one of the intermediate data tensors must have a smaller size than the input and output tensors

Basic capability of any AE:

Dimensionality reduction, or compression of data into smaller space, or extraction of essential features.

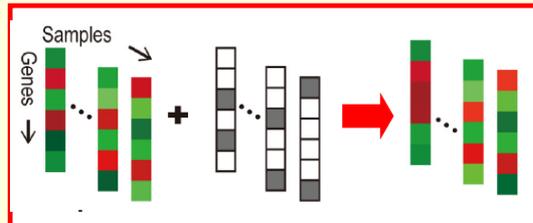
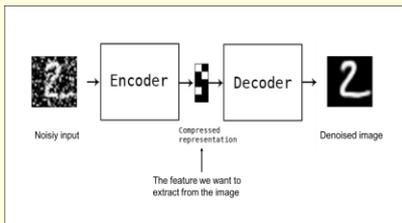


Examples:

Denoising autoencoder

Image denoising

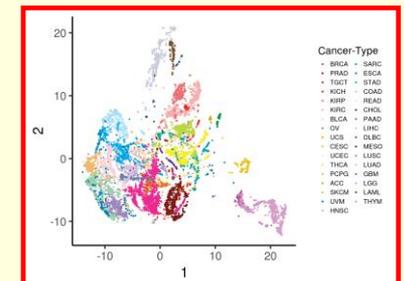
ADAGE: analysis using denoising autoencoders of gene expression



Variational autoencoder

Generating images

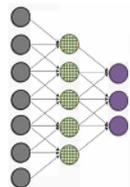
Tybal: classification of cancer samples based on gene expression



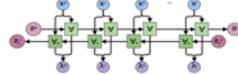
Examples overview

#	1	2	3	4	5
Biological Appliation	Bioimage segmentation/ fly brain connectome project	Genomics/ predicting the function of <u>non-coding DNA</u> (~98%)	Genomics/ clustering of cancer samples based on <u>gene expression</u> (~2%)	Bioimage synthesis / developmental biology	Small drug molecule design
Network type	Convolutional Neural Network	Recurrent Neural Network	Auto-encoder	Generative Adversarial Network	Reinforcement Learning Network
ML type	Supervised	Supervised	Unsupervised	Unsupervised	Reinforcement

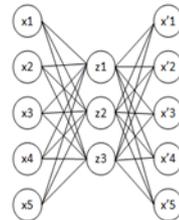
CNN



RNN



AE



How #3 differs from #1 and #2:

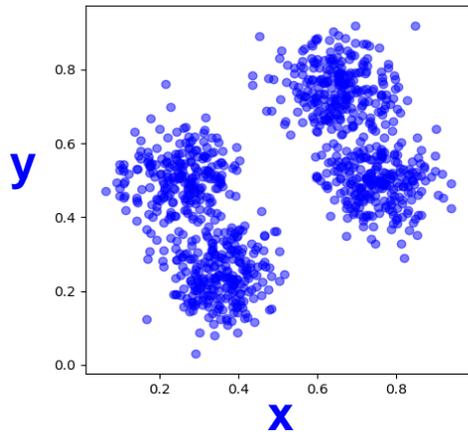
- 1) unsupervised ML approach
- 2) no specialized “autoencoder” layer
- 3) a composite network that comprises two subnetworks

Basic autoencoders: a simple example

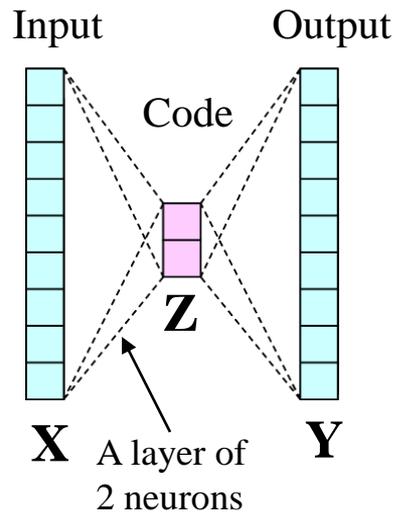
tensors, units, layers, parameters, activations, hyperparameters, deep network



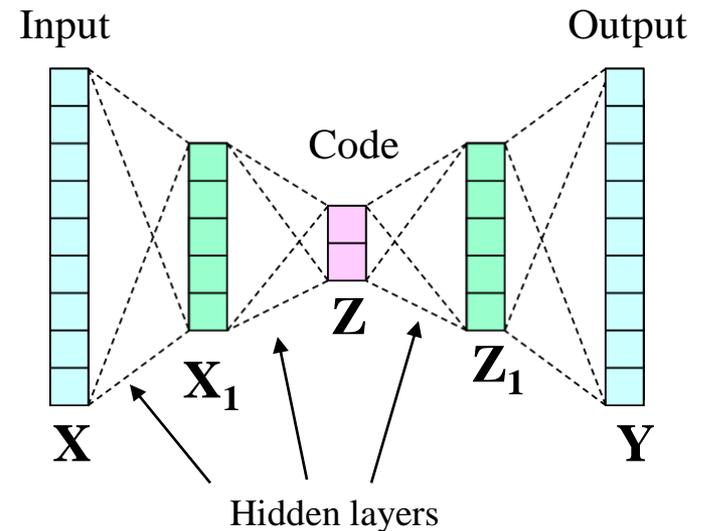
Synthetic data
(only 2 out of 9 components
are shown)



Model 1 (shallow)



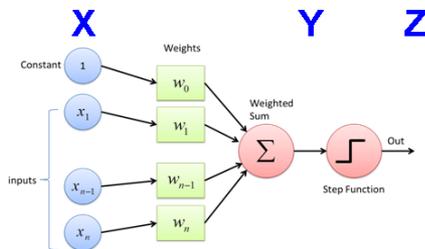
Model 2 (deep)



One neuron:
 $Z = A(\sum w_i \cdot X_i + b)$
(output = scalar)

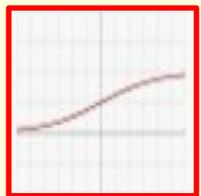
Hyperparameters:

- types of the layers:
Convolutional (for image data), or
LSTM-based (for sequence data), or
Dense/Fully Connected (to be used here)
- total # layers: **2** (Model 1) or **4** (Model 2)



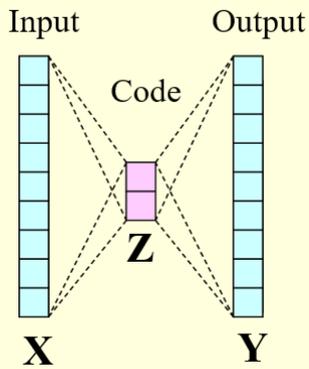
A layer of neurons:
 $Z = A(W \cdot X + b)$
(output = vector)

A(Y)



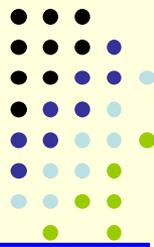
- activations: **linear** (Model 1) or **sigmoid** (Model 2)
- size of the latent space **Z: 2**
- sizes of other tensors (X, Y: **9**; X1, Z1: **5**)

Deep network: **>= 2 hidden layers**



The training code for basic Model 1

backend, encoder, code, decoder, combined_model, compile, loss, optimizer, fit, checkpoint, epoch, callback



Header:

- general Python imports
- Numpy import
- Keras library imports

Getting data

- independent random variables
- degrees of freedom

Defining a model

- encoder, code, decoder
- combined_model
- loss, optimizer, compile

Running the model

- checkpoint, fit, epoch, callback

Backend: a deep learning framework that provides a low-level support for Keras; by default = Tensorflow

```

Select denisovga@biowulf:/data/denisovga/

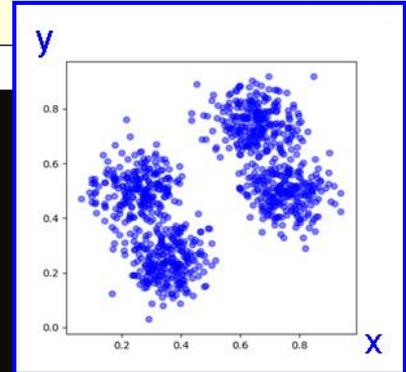
# Imports
import os
import numpy as np
from keras.layers import Dense
from keras.models import Input, Model
from keras.callbacks import ModelCheckpoint

# Get data
num_rows = 1000
np.random.seed(1)
rint = np.random.randint(0, high=4, size=(num_rows, 2))
coordx = np.array([0.35, 0.25, 0.75, 0.65])
coordy = np.array([0.25, 0.5, 0.50, 0.75])
eps = np.random.normal(scale=0.07, size=(num_rows, 2))
x = coordx[rint[:, 0]] + eps[:, 0]
y = coordy[rint[:, 0]] + eps[:, 1]
x_train = np.transpose(np.array([x, y, x*x, x*y, y*y, \
                                  x*x*x, x*x*y, x*y*y, y*y*y]))

# Define the ae model
X = Input((9,))
Z = Dense(2)(X)
encoder = Model(X,Z, name = 'encoder')
Code = Input(shape=(2,))
Y = Dense(9)(Code)
decoder = Model(Code, Y, name = 'decoder')
combined_model = Model(inputs = X, outputs = decoder(encoder(X)))

combined_model.compile(loss='mse', optimizer='sgd')

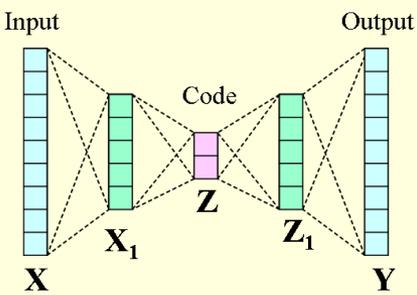
# Run the model on the data
checkpointer = ModelCheckpoint(filepath="basic_ae_model1.h5")
combined_model.fit(x_train, x_train, batch_size=32, epochs=500, \
                  callbacks=[checkpointer])
    
```



x	y	xx	xy	yy	xxx	xyy	yyy
...
...
...
...
...
...
...
...
...
...

The prediction code for basic Model 2

encoder, decoder, load_weights, predict



Header:

- Numpy import
- Keras library imports
- **Sequential**

Getting data

- independent random variables
- degrees of freedom

Defining a model

- **encoder**, code, **decoder**
- combined_model
- loss, optimizer, compile

Running the model

- **load_weights**
- **predict**

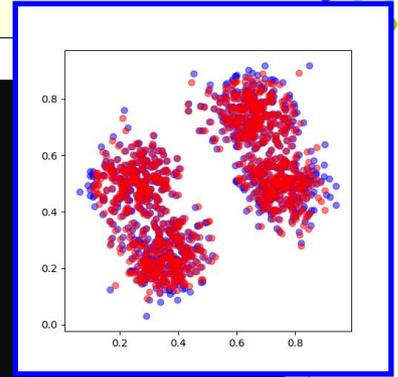
```
Select denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro
#!/usr/bin/env python

# Imports
import os
import numpy as np
from keras.layers import Dense
from keras.models import Sequential
import matplotlib.pyplot as plt

# Get data
num_rows = 1000
np.random.seed(2)
rint = np.random.randint(0, high=4, size=(num_rows, 2))
coordx = np.array([0.35, 0.25, 0.75, 0.65])
coordy = np.array([0.25, 0.5, 0.5, 0.75])
eps = np.random.normal(scale=0.07, size=(num_rows, 2))
x = coordx[rint[:, 0]] + eps[:, 0]
y = coordy[rint[:, 0]] + eps[:, 1]
x_train = np.transpose(np.array([x, y, x*x, x*y, y*y, \
                                 x*x*x, x*x*y, x*y*y, y*y*y]))

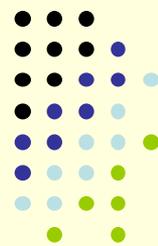
# Define the model
encoder = Sequential()
encoder.add(Dense(5, activation = 'sigmoid', input_shape=(9,)))
encoder.add(Dense(2, activation = 'sigmoid'))
decoder = Sequential()
decoder.add(Dense(5, activation = 'sigmoid', input_shape=(2,)))
decoder.add(Dense(9, activation = 'sigmoid'))
combined_model = Sequential()
combined_model.add(encoder)
combined_model.add(decoder)
combined_model.compile(loss='mse', optimizer='adam')

# Run the model on the data
combined_model.load_weights("basic_ae_model2_h5")
x_recon = decoder.predict(encoder.predict(x_train))
plt.figure(figsize=(5, 5))
plt.scatter(x_train[:, 0], x_train[:, 1], alpha=0.5, color='blue')
plt.scatter(x_recon[:, 0], x_recon[:, 1], alpha=0.5, color='red')
plt.savefig("basic_ae_model2.png")
plt.show()
```

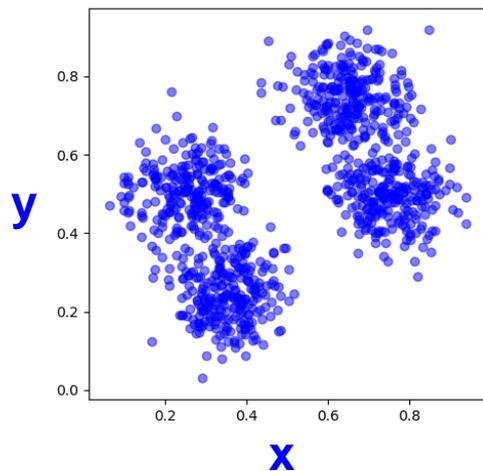


Summary: basic autoencoders vs PCA

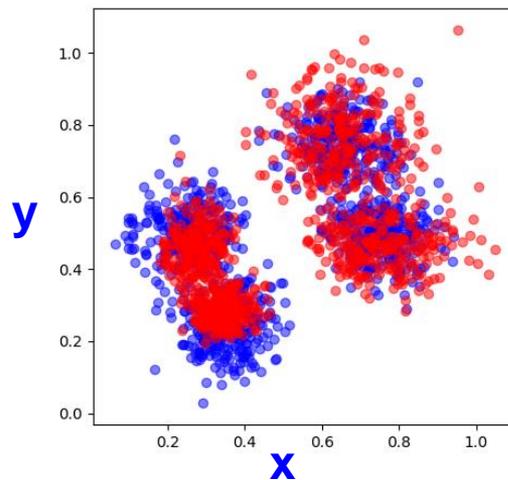
https://www.cs.toronto.edu/~urtasun/courses/CSC411/14_pca.pdf



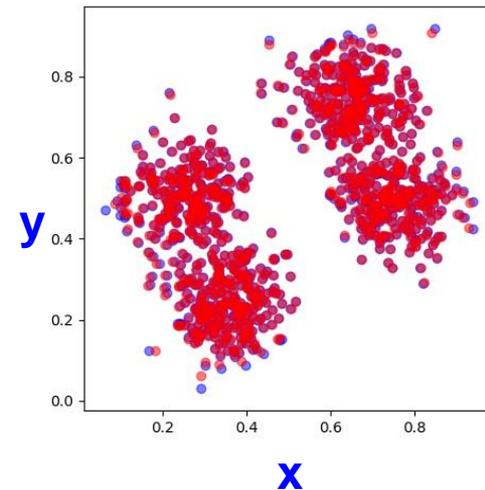
Training data



Model 1 (~PCA)
reconstruction
(after 3,000 epochs)



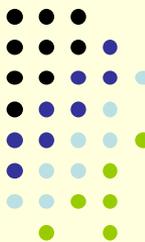
Model 2
reconstruction
(after 3,000 epochs)



Conclusions:

- 1) Model 1, which mimics the PCA, cannot capture the nonlinear relationships between the data components
- 2) Model 2, the deep autoencoder with nonlinear activations, supersedes Model 1 and can be regarded as a nonlinear extension of the PCA.

How to run basic autoencoders on Biowulf?



```
Select denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro
sinteractive --gres=gpu:p100:1 --mem=4g
module load DLBio/class3
...
ls $DLBIO_BIN
basic_ae_model1_predict.py  basic_ae_model2_predict.py
basic_ae_model1_train.py   basic_ae_model2_train.py

basic_ae_model1_train.py
Using TensorFlow backend.
...
Epoch 1/3000
1000/1000 [=====] - 2s 2ms/step - loss: 0.1292
Epoch 2/3000
1000/1000 [=====] - 0s 91us/step - loss: 0.0981
...

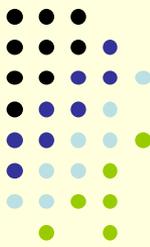
basic_ae_model1_predict.py
...

basic_ae_model2_train.py
...

basic_ae_model2_predict.py
...

1,39 | All
```

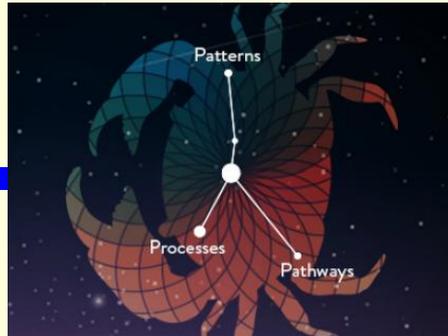
Example 3. Tybalt: extracting a biologically relevant latent space from cancer transcriptomes



Tybalt: J.P.Way, C.S.Greene, Pacif Symp. on Biocomputing (2018)

TCGA: J.N.Weinstein et al, Nature Genetics 45 (2013)

<https://hpc.nih.gov/apps/Tybalt.html>



Input:
gene expression profiles for:
- 33 types of cancer
- 9,732 tumor samples
- 727 normal samples

Tasks:
1) reduce dimensionality of the feature space: 5000 → 100
2) using the essential features, classify / cluster the samples into 34 groups

Data from:

The Cancer Genome Atlas (TCGA)

- NIH program led by NCI and NHGRI

download_data.sh

Raw data
(TSV)

process_data.py

Pre-processed
data (TSV)

Checkpoints
(HDF5)

Encoded data;
tSNE features
(TSV)

Plots
(PNG)

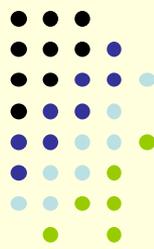
tybalt_train.py

tybalt_predict.py

tybalt_visualize.R

deep learning

Overview of the Tybalt training code (only the main function is shown)



<https://hpc.nih.gov/apps/Tybalt.html>

↑
Imports statements,
other function
definitions

Header

- parse the command line options

Getting data

- data in TSV format

Defining a model

- ADAGE model
- VAE model
- Lambda layer
- two loss functions used by the VAE model

Running the model

- fit
- predict
- perform_tSNE
- more on GD-based optimization

```
Select denisovga@biowulf:/data/denisovga/Tybalt
if __name__ == '__main__':
    opt, checkpoint_combined, checkpoint_encoder, checkpoint_decoder = \
        parse_command_line_arguments("train")

    # Load data
    rnaseq_df, rnaseq_train_df, rnaseq_test_df, original_dim, latent_dim, \
        hidden_dim, beta = get_data()

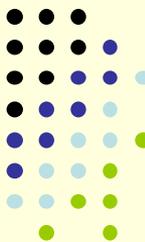
    # Define model
    encoder, decoder, combined_model = build_combined_model(opt.model_name, \
        original_dim, latent_dim, hidden_dim, int(opt.depth), beta, \
        opt.noise, opt.sparsity)
    combined_model.summary()
    if opt.model_name == "vae":
        combined_model.compile(optimizer='adam', loss=None, loss_weights=[beta])
    elif opt.model_name == "adage":
        combined_model.compile(optimizer='adam', loss='mse')

    # Fit the AE model
    print("checkpoint_combined=", checkpoint_combined)
    checkpointer = ModelCheckpoint(filepath=checkpoint_combined, \
        verbose=opt.verbose, save_weights_only=True)

    if opt.model_name == "vae":
        combined_model.fit(rnaseq_train_df, shuffle=True, \
            epochs=opt.num_epochs, batch_size=opt.batch_size, \
            validation_data=(rnaseq_test_df, None), \
            callbacks=[warmUpCallback(beta, opt.kappa), checkpointer])
    elif opt.model_name == "adage":
        combined_model.fit(rnaseq_train_df, rnaseq_train_df, shuffle=True, \
            epochs=opt.num_epochs, batch_size=opt.batch_size, \
            validation_data=(rnaseq_test_df, rnaseq_test_df), \
            callbacks=[checkpointer])

    combined_model.save_weights(checkpoint_combined)
    encoder.save_weights(checkpoint_encoder)
    decoder.save_weights(checkpoint_decoder)
```

Tybalt data



(RNAseq gene expression, copy number, mutation and clinical)

Glioblastoma NF1 data:

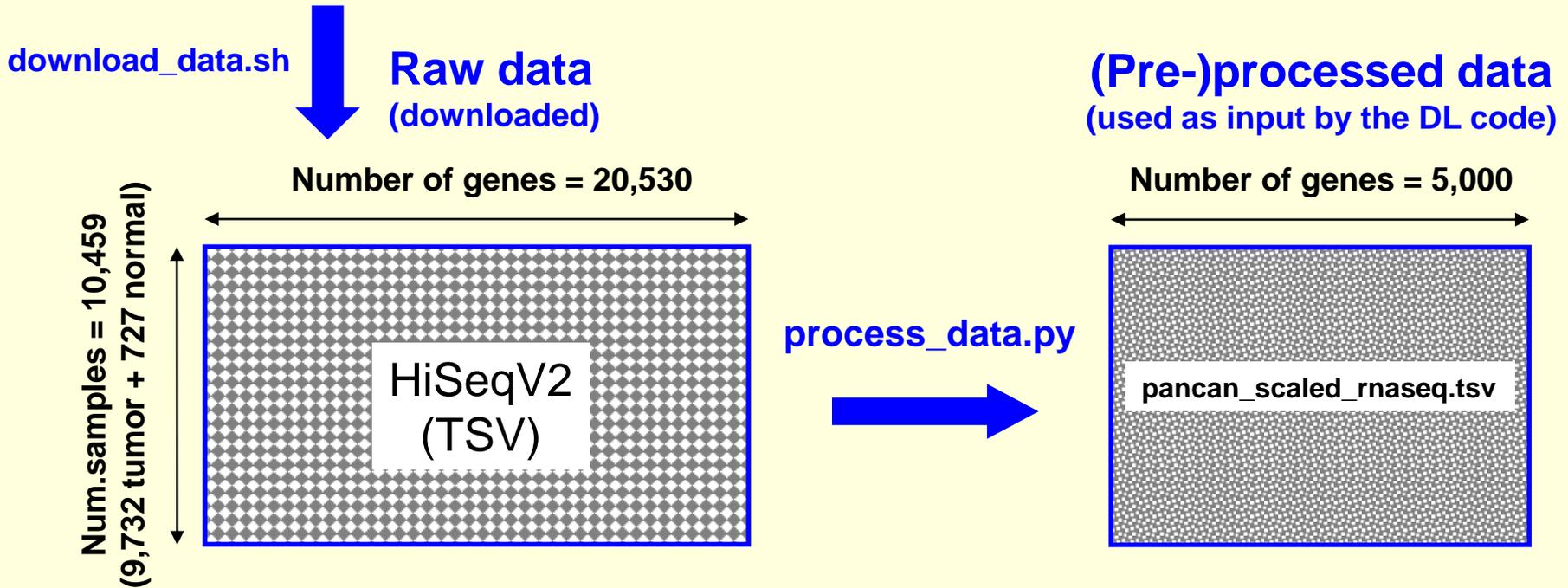
<https://zenodo.org/record/56735/#.XPevDFVKhhE>

UCSC Xena Data Browser Copy Number Data:

<https://zenodo.org/record/827323#.XPexAFVKhhE>

Clinical data files from JHU:

<http://snaptron.cs.jhu.edu/data/tcga/samples.tsv>



Other raw data

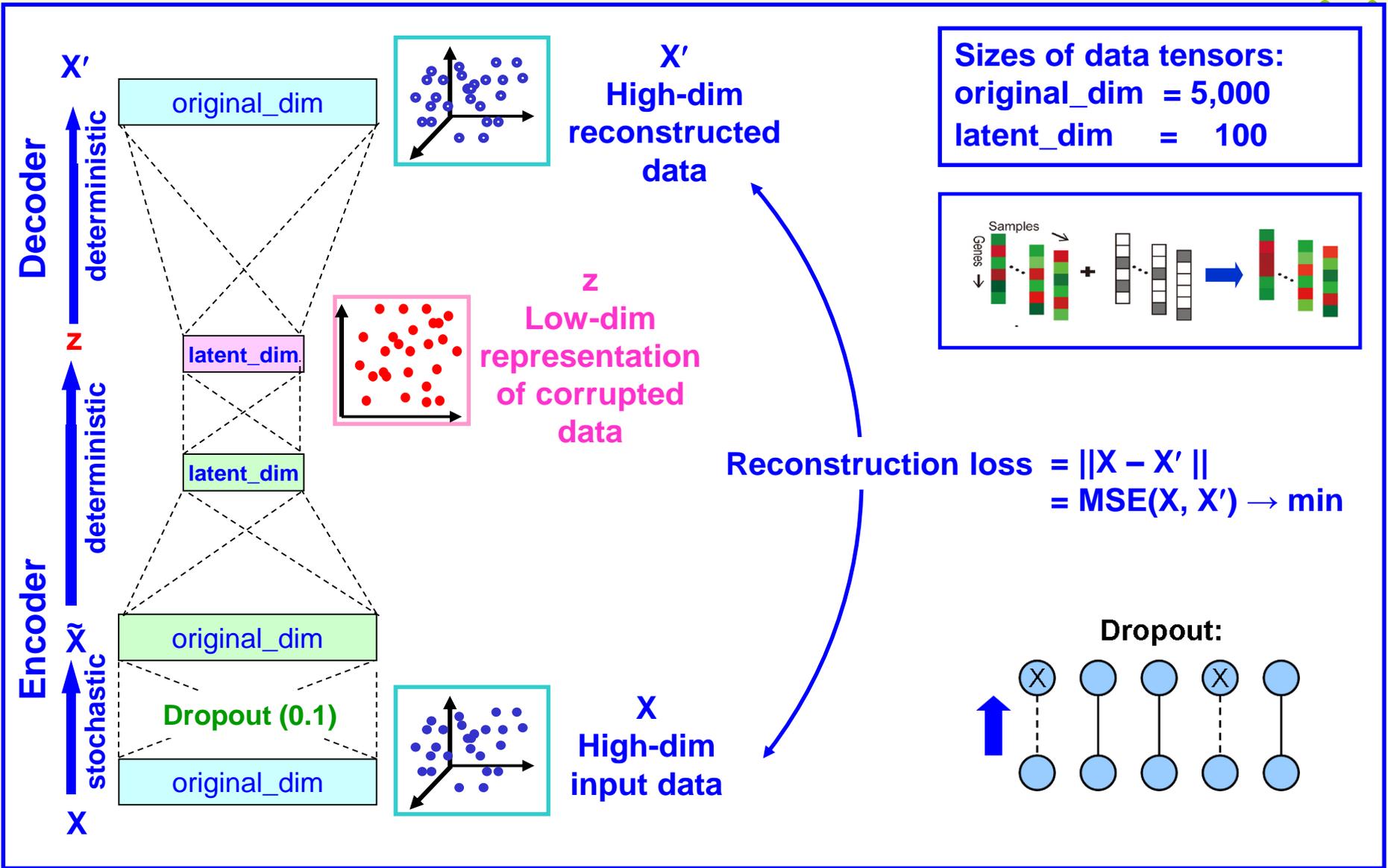
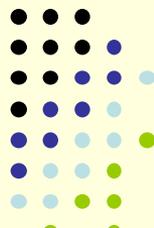
	Shape
Gistic2_CopyNumber_all_thresholded.by_genes	(24776, 10845)
PANCAN_mutation	(2034801, 10)
samples.tsv	(11284, 860)
PANCAN_clinicalMatrix	(12088, 35)

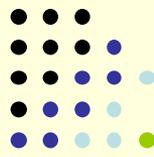
Other processed data

	Shape
pancan_mutation.tsv	(7515, 29829)
status_matrix.tsv	(7230, 29829)
tybalt_features_with_clinical.tsv	(10375, 117)
...	

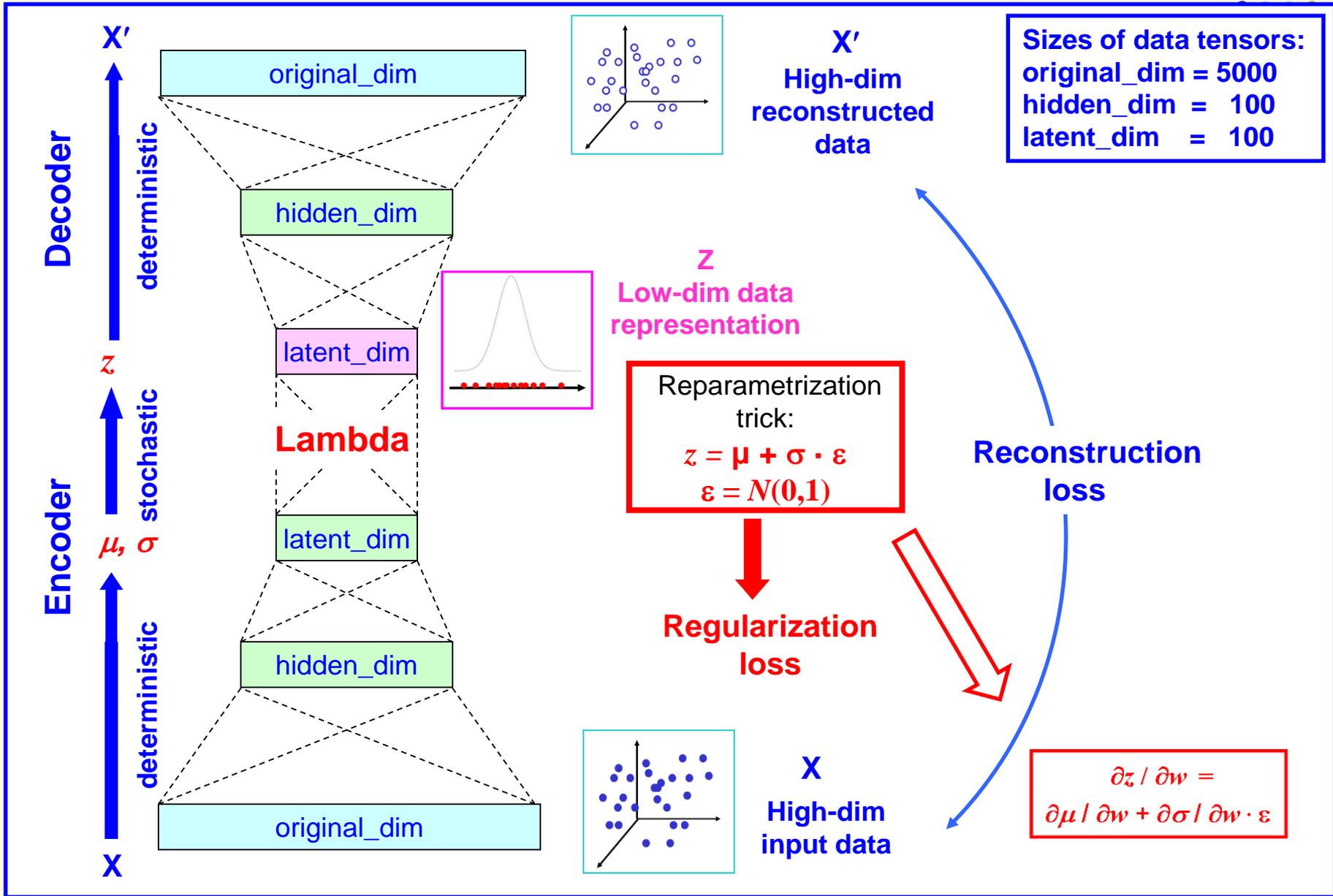
The ADAGE (denoising autoencoder) model

ADAGE paper: J.Tan et al., mSystems (2016)

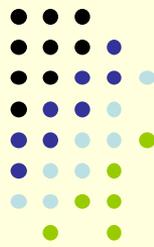




The VAE (variational autoencoder) model



tSNE: t-Distributed Stochastic Neighbor Embedding



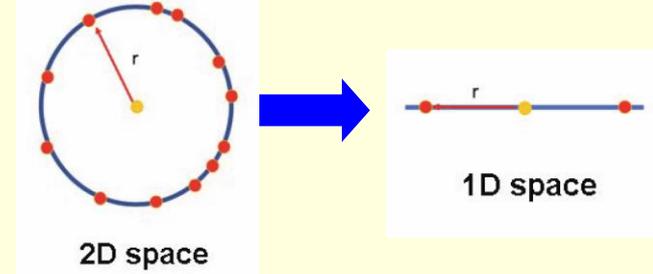
L. Van der Maaten, J.Hinton – J. Machine Learning Res. 9 (2008) 2579-2605

<https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>

Task: map data points, together with their neighbors, from a high-dim to a low-dim space, for subsequent visualization

Problems:

- 1) Simple projection **does not preserve clusters**
- 2) The **curse of dimensionality:** mapped **datapoints** have tendency to **get crowded** or merged



Solution:

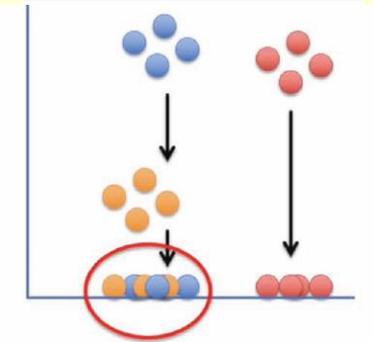
- 1) define a neighborhood for each point probabilistically
- 2) map/embed all the neighborhoods from the high-dim to the low-dim space
- 3) minimize the KL divergence between the GD and StD using stochastic gradient descent

For high-dim space:
Gaussian distribution (GD)

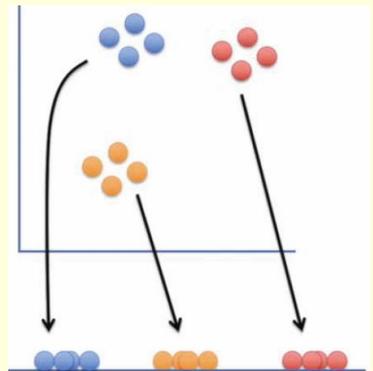
For low-dim space:
Student **t-distribution (StD)**

$$P_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_k - x_i\|^2 / 2\sigma_i^2)}$$

$$Q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_k - y_i\|^2)^{-1}}$$

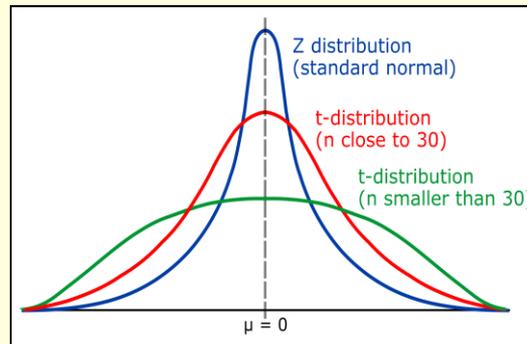


tSNE



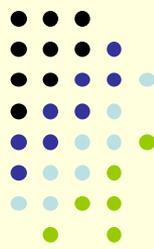
“High”-dim (=2D) space

“Low”-dim (=1D) space



Since the t-distribution has a longer tail than the Gaussian distribution, “stretching” of an effective neighborhood during the mapping allows to “resolve” the crowded points.

How to run the Tybalt application on Biowulf?



<https://hpc.nih.gov/apps/tybalt.html>

<https://github.com/greenelab/tybalt>

```
Select denisovga@biowulf:/data/denisovga/1_DL_Course/3_AEs
sinteractive --mem=16g \
             --gres=gpu:p100:1,lscratch:10

module load tybalt

ls $TYBSALT_SRC
download_data.sh  __pycache__
models.py         tybalt_predict.py
options.py        tybalt_train.py
process_data.py   tybalt_visualize.R

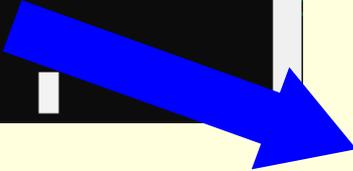
download_data.sh

process_data.py

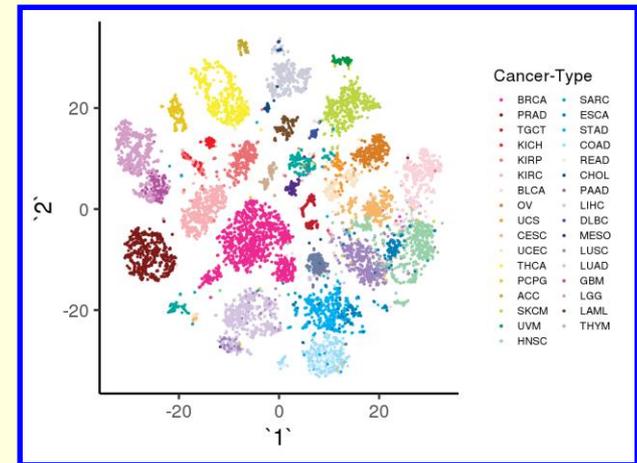
tybalt_train.py   -m vae
tybalt_predict.py -m vae
tybalt_visualize.R -v

tybalt_train.py   -m adage
tybalt_predict.py -m adage
tybalt_visualize.R -a

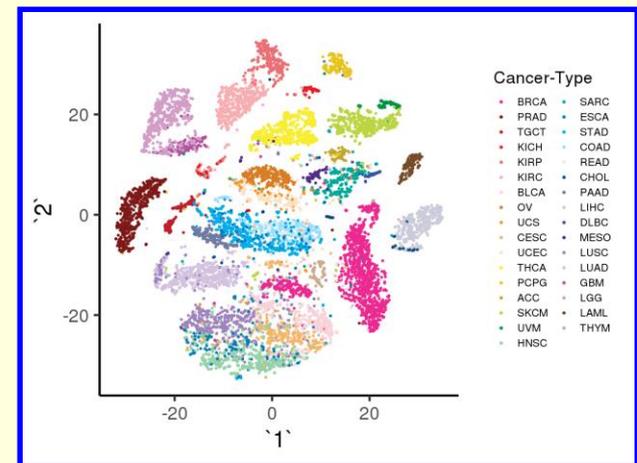
22,39
```



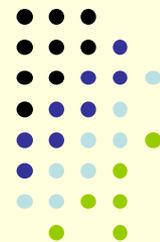
VAE model + tSNE



AGAGE model + tSNE



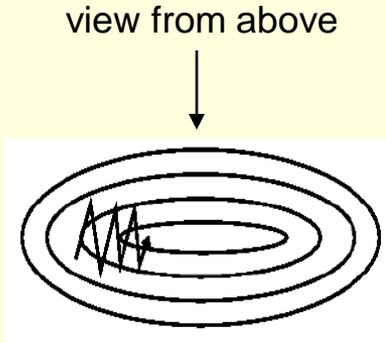
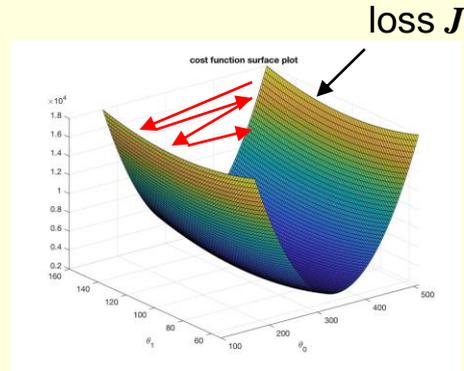
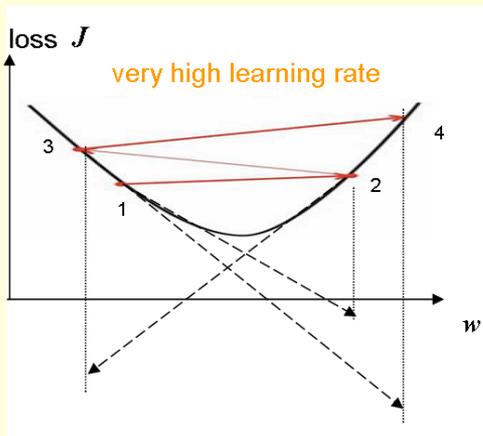
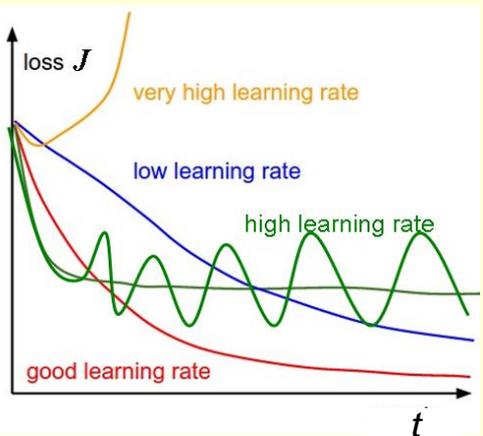
More on the gradient descent-based optimizers: momentum and Nesterov accelerated gradient



<http://ruder.io/optimizing-gradient-descent>

$$\Delta w_t = -\gamma \cdot \nabla_w J(w_t)$$

w = vector of weights; t = update #; γ = learning rate;
 J = loss function; $\Delta w_t = w_{t+1} - w_t$



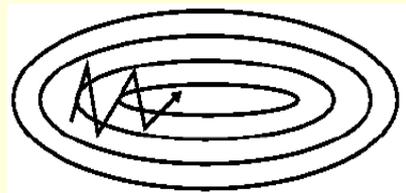
- small γ \rightarrow slow convergence along the valley
- larger γ \rightarrow oscillations in the perpendicular dir.

$$\Delta w_t = \mu \cdot \Delta w_{t-1} - \gamma \cdot \nabla_w J(w_t)$$

- gradient descent formula with **momentum μ** (usually, = 0.9)

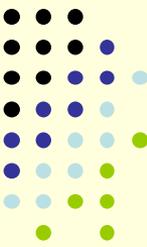
$$\Delta w_t = \mu \cdot \Delta w_{t-1} - \gamma \cdot \nabla_w J(w_t - \mu \cdot \Delta w_{t-1})$$

- gradient descent formula with momentum μ and **Nesterov accelerated gradient**



`keras.optimizers.SGD(lr=0.01, momentum=0.0, nesterov=False)`

Conclusions



1) Intro using a simple example

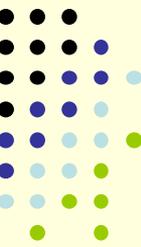
- basic **autoencoder with single hidden layer** mimics the **PCA** and cannot capture the nonlinear relationships between data components
- **deep basic autoencoder** with nonlinear activations supercedes the **PCA** and can be regarded as **nonlinear extension of the PCA**

2) The Tybalt application:

- **ADAGE** and **VAE** models
- **VAE: reparametrization** trick
- **VAE: reconstruction** and **regularization** losses
- **tSNE** for visualization of clusters

3) Other topics:

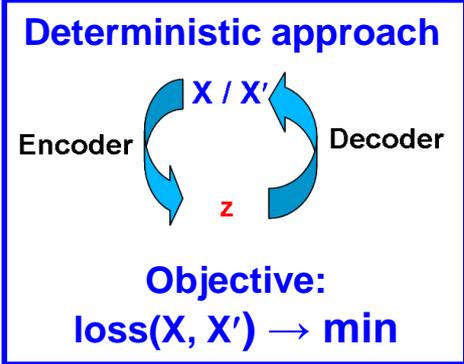
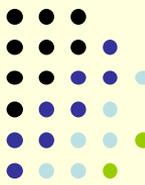
- gradient descent-based optimization algorithms:
Momentum and **Nesterov Accelerated Gradient**



BACKUP SLIDES

Why “Variational”? Computing the VAE loss.

<https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/>



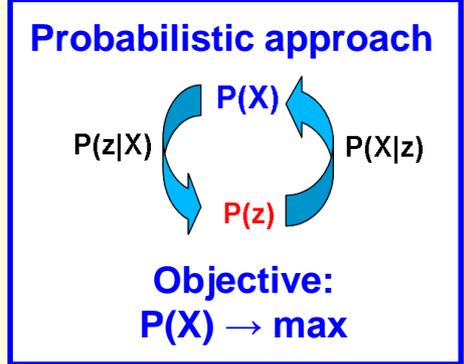
$$\log P(X) = \log \int P(X|z) P(z) dz \rightarrow \max$$

“evidence”

apply the Bayes rule; replace integration with sampling

$$\log P(X) = E_{z \sim P(z|X)} P(X|z) - D_{KL}[P(X|z) || P(z)] \rightarrow \max$$

add and subtract an approximate distribution $N(\mu(X), \sigma(X))$; make other adjustments



$$\log P(X) - D_{KL}[N(\mu(X), \sigma(X)) || P(z|X)] = E_{z \sim N(\mu, \sigma)} P(X|z) - D_{KL}[N(\mu(X), \sigma(X)) || N(0,1)] \rightarrow \max$$

- variational inference



“evidence lower bound” (ELBO)

Keras implementation

adjustable weight

$$\text{loss} = -\text{ELBO} \approx \underbrace{\text{BC}(X, X')}_{\text{Reconstruction loss}} - \beta \cdot \sum [\underbrace{\sigma_i(X) + \mu_i(X)^2 - 1 - \log \sigma_i(X)}_{\text{Regularization loss}}] \rightarrow \min$$

D_{KL} = Kullback-Leibler divergence:
 $D_{KL}[p(x) || q(x)] = \int p(x) \cdot \ln[p(x)/q(x)]$

BC = Binary cross-entropy