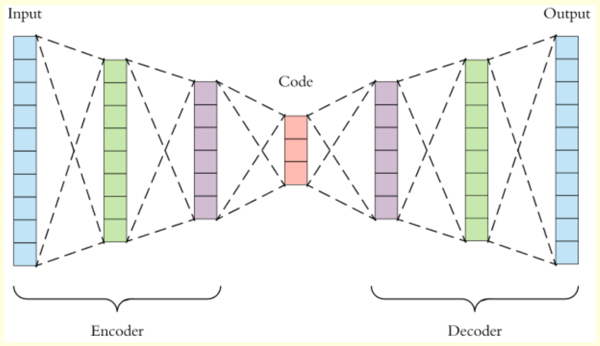
Deep Learning by Example on Biowulf.

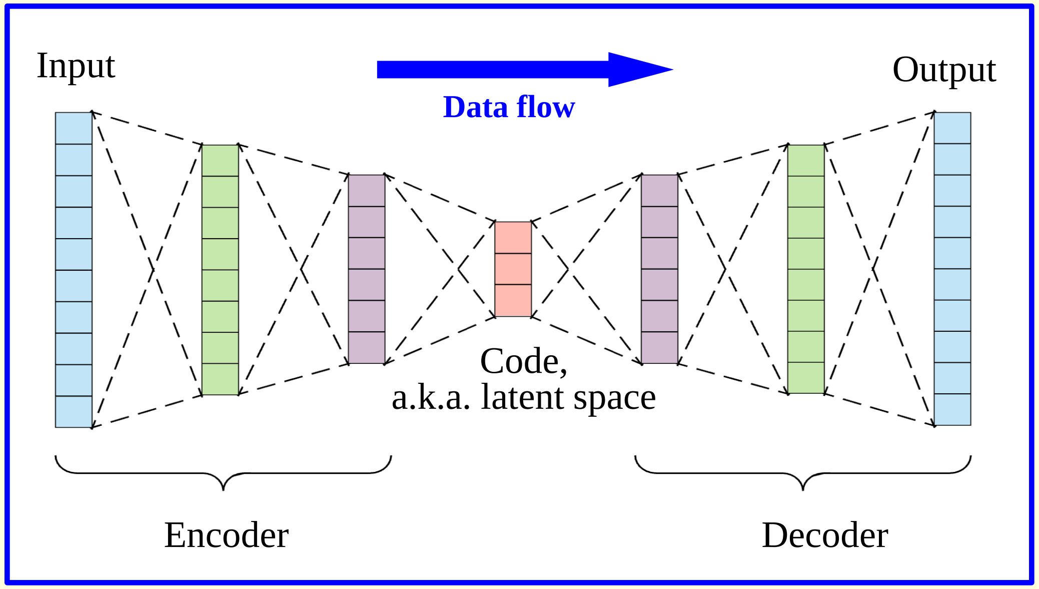
Class #3:
Autoencoders, hyperparameter optimization and their application to reduction of dimensionality of cancer transcriptome.

Gennady Denisov, PhD



Intro and goals

encoder, decoder, code / latent space



What is autoencoder?

Two basic requirements:

1) The dimensions of the input and output tensor must be the same
2) At least one of the intermediate data tensors must have a smaller dimension than the input and output tensors

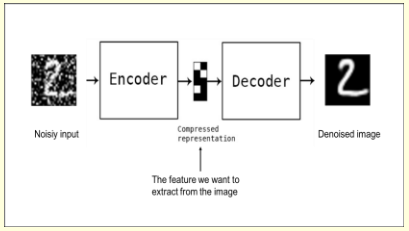
Basic capability of any AE:

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

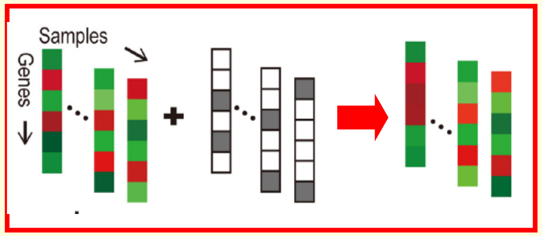
Examples:

Denoising autoencoder

Image denoising


Encoder
Decoder
Noisy input
Compressed representation
Denoised image
The feature we want to extract from the image

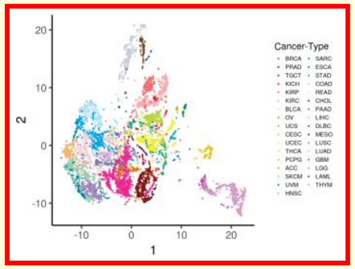
ADAGE: analysis using denoising autoencoders of gene expression



Variational autoencoder

Generating images

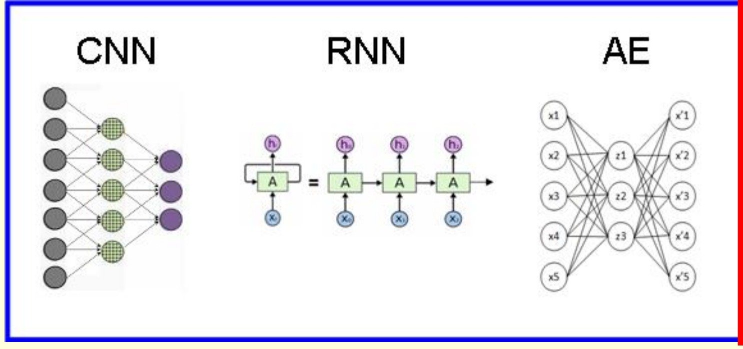
Tybalt: reduction of dimensionality of a cancer transcriptome



Hyperparameter optimization (HPO): KerasTuner, CANDLE

Examples overview

Class #	1	2	3	4	5	6	7
Bio app	Bioimage segmentation / fly brain connectome	Genomics / prediction of function of non-coding DNA	Genomics / reduction of dimensionality of cancer transcriptome	Bioimage synthesis / developmental biology	Drug molecule design	Genomics/ classification of cancer types	Chemical Compound / drug property prediction
Neural network	Convolutional	Recurrent or 1D-Convolutional	Autoencoder	Generative Adversarial	Reinforcement Learning	Graph Convolutional	Message Passing (Graph)
ML type	Supervised	Supervised	Unsupervised	Unsupervised	Reinforcement	Supervised	Supervised



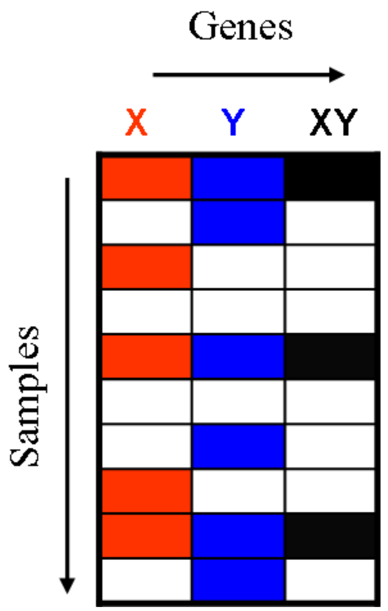
How #3 differs from #1 and #2:

1) unsupervised ML approach
2) there is no autoencoder-specific type of layer that would be used as a building block
3) a composite network comprising 2 subnetworks
4) will discuss hyperparameter optimization

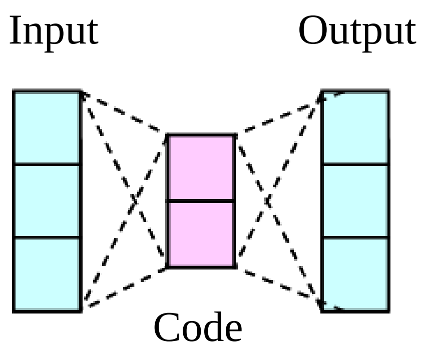
Basic autoencoder models

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

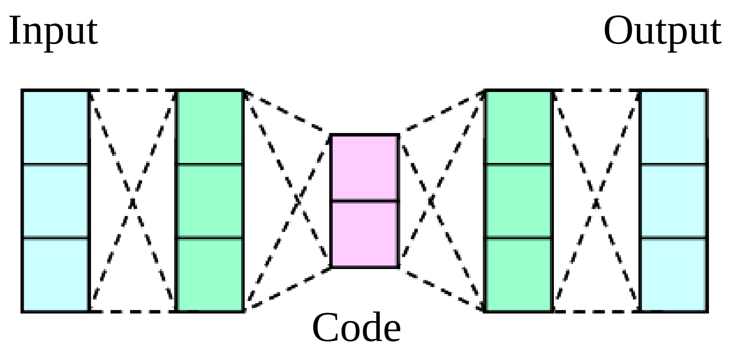
Gene expression data matrix



Shallow model



Deep model



Input: a gene expression data matrix with three columns (=genes), a number of rows (=samples) and the values 1 (=gene is expressed) or 0 (=gene is unexpressed). Two of the genes are expressed independently, whereas the 3rd gene is expressed if and only if the first two genes are both expressed.

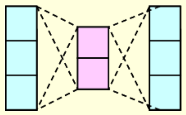
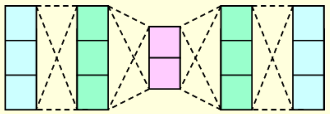
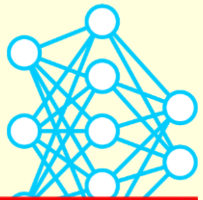
Task: train the basic autoencoder models with code size = 2 on this data.

Deep network / model: >= 2 hidden layers with adjustable parameters

Hyperparameters:

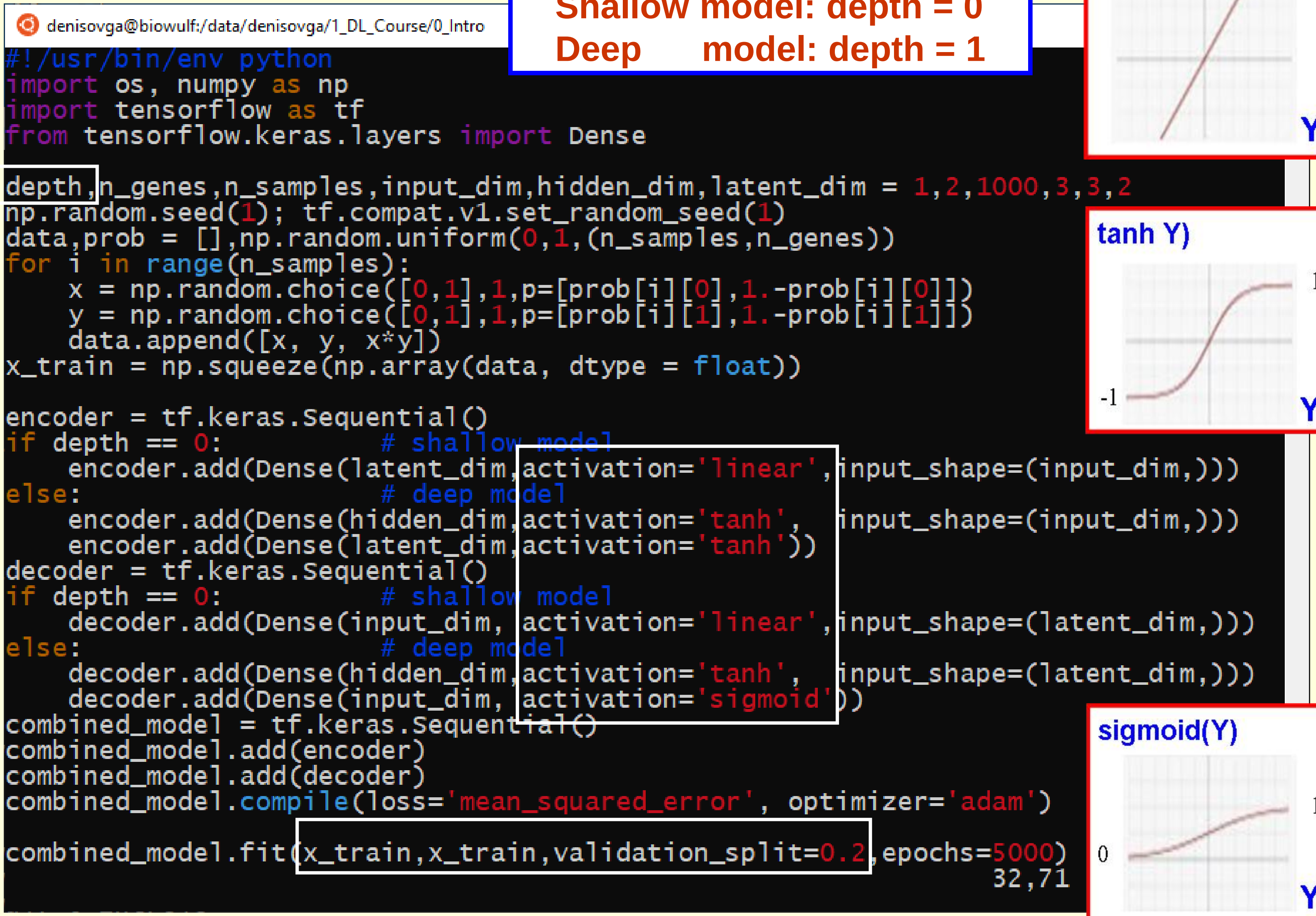
- **types of the layers**: **Dense/Fully Connected**
- **depth** of encoder and decoder, i.e. the # of hidden ("green") tensors: **0** (Shallow model) or **1** (Deep model)
- size of the code tensor ("**latent_dim**"): **2**
- size of input/output tensors ("**input_dim**"): **3**
- **activations**: **linear** (Shallow model) or **tanh/sigmoid** (Deep model)

Training code for the basic models

encoder model, decoder model, combined model, validation loss

Shallow model: depth = 0
Deep   model: depth = 1



1) Header

```
#!/usr/bin/env python
import os, numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense
```

2) Get data

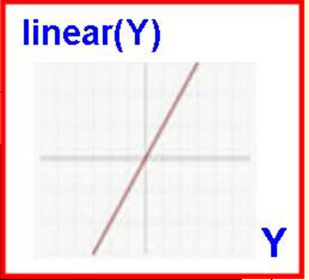
```
depth,n_genes,n_samples,input_dim,hidden_dim,latent_dim = 1,2,1000,3,3,2
np.random.seed(1); tf.compat.v1.set_random_seed(1)
data,prob = [],np.random.uniform(0,1,(n_samples,n_genes))
for i in range(n_samples):
    x = np.random.choice([0,1],1,p=[prob[i][0],1.-prob[i][0]])
    y = np.random.choice([0,1],1,p=[prob[i][1],1.-prob[i][1]])
    data.append([x, y, x*y])
x_train = np.squeeze(np.array(data, dtype = float))
```

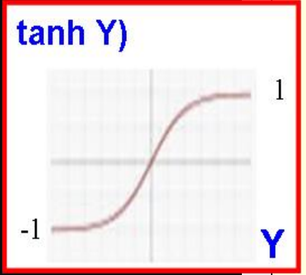
3) Define a model

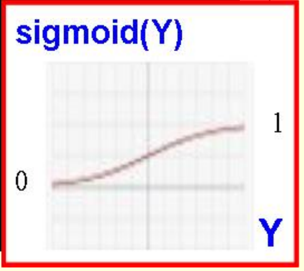
```
encoder = tf.keras.Sequential()
if depth == 0:           # shallow model
    encoder.add(Dense(latent_dim,activation='linear',input_shape=(input_dim,)))
else:                    # deep model
    encoder.add(Dense(hidden_dim,activation='tanh',  input_shape=(input_dim,)))
    encoder.add(Dense(latent_dim,activation='tanh'))
decoder = tf.keras.Sequential()
if depth == 0:           # shallow model
    decoder.add(Dense(input_dim, activation='linear',input_shape=(latent_dim,)))
else:                    # deep model
    decoder.add(Dense(hidden_dim,activation='tanh',  input_shape=(latent_dim,)))
    decoder.add(Dense(input_dim, activation='sigmoid'))
combined_model = tf.keras.Sequential()
combined_model.add(encoder)
combined_model.add(decoder)
combined_model.compile(loss='mean_squared_error', optimizer='adam')
```

4) Run the model

```
combined_model.fit(x_train,x_train,validation_split=0.2,epochs=5000)
                                                                 32,71
```

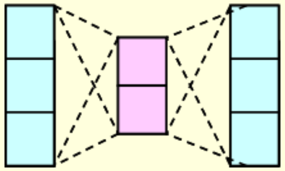


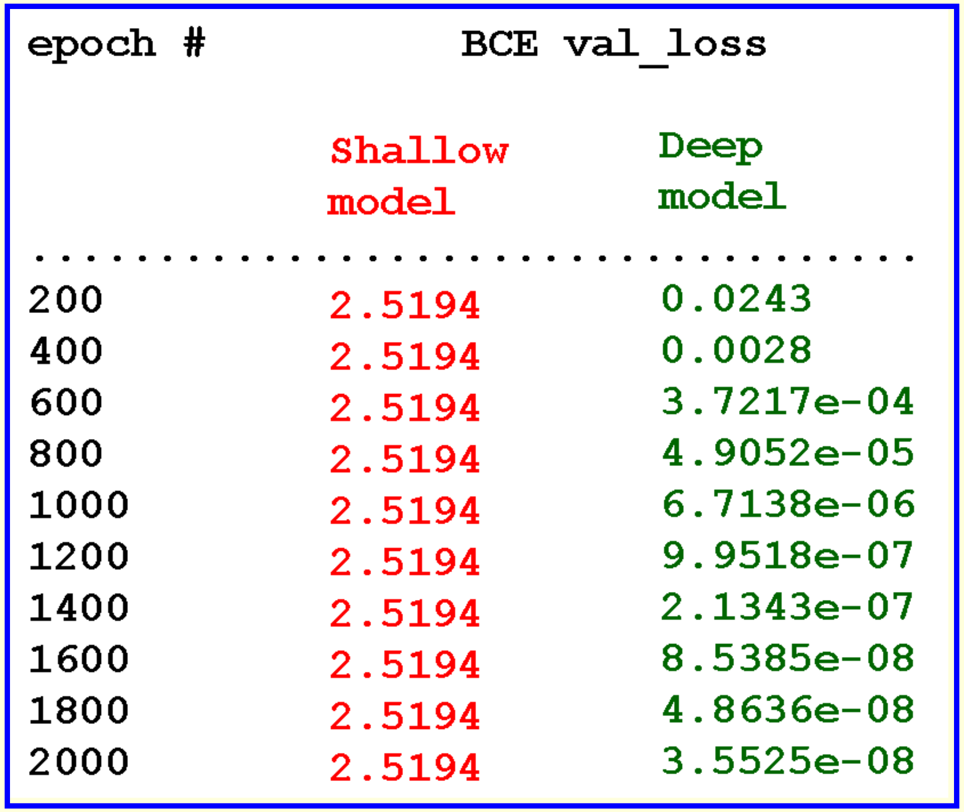




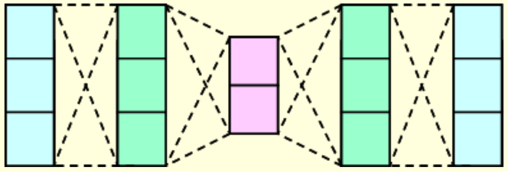
Results for the basic models: deep autoencoder vs PCA

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





epoch #	BCE val_loss	
	Shallow model	Deep model
200	2.5194	0.0243
400	2.5194	0.0028
600	2.5194	3.7217e-04
800	2.5194	4.9052e-05
1000	2.5194	6.7138e-06
1200	2.5194	9.9518e-07
1400	2.5194	2.1343e-07
1600	2.5194	8.5385e-08
1800	2.5194	4.8636e-08
2000	2.5194	3.5525e-08

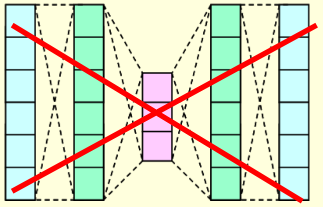


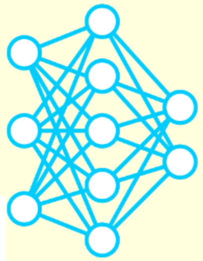
Conclusions:

1) The shallow model with linear activation, which is known to mimic the PCA, (see the link above) cannot capture the nonlinear relationships between variables / decouple them
2) The deep model with nonlinear activations supersedes the shallow model and can be regarded as a nonlinear extension of the PCA.

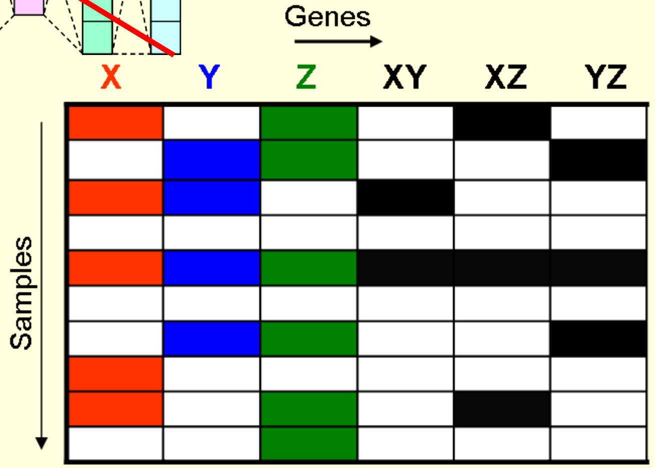
HP optimization with KerasTuner (v1.0.3)

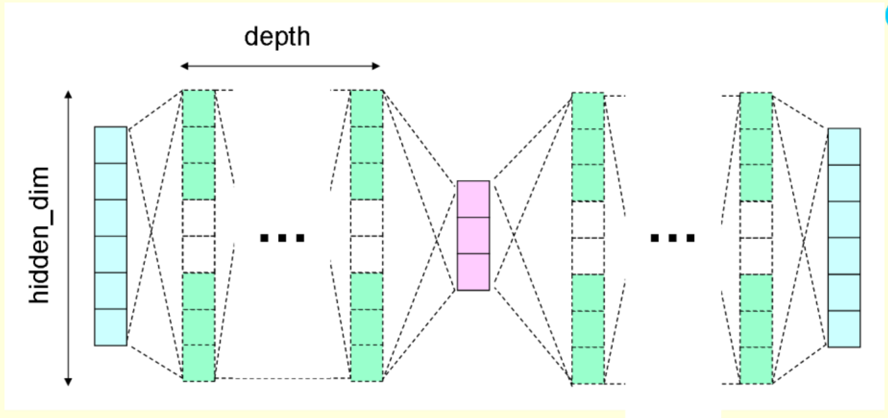
https://keras-team.github.io/keras-tuner/



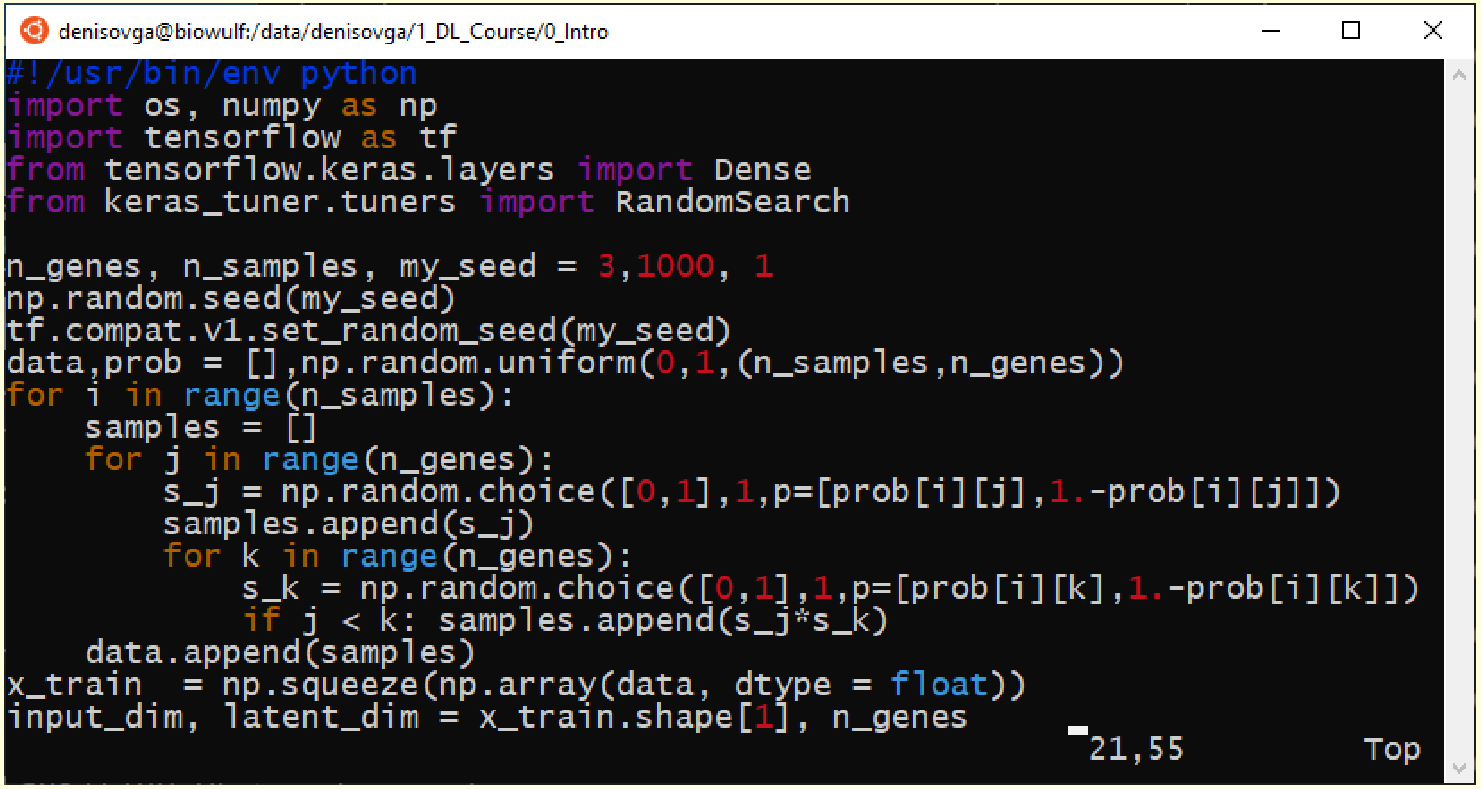


Data matrix





Model

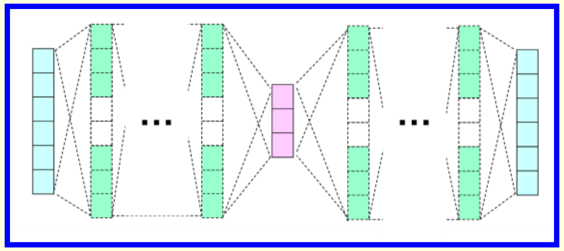


1) Header

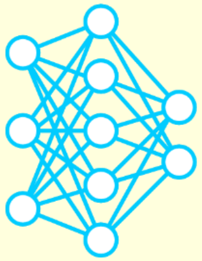
```
#!/usr/bin/env python
import os, numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense
from keras_tuner.tuners import RandomSearch
```

2) Get data

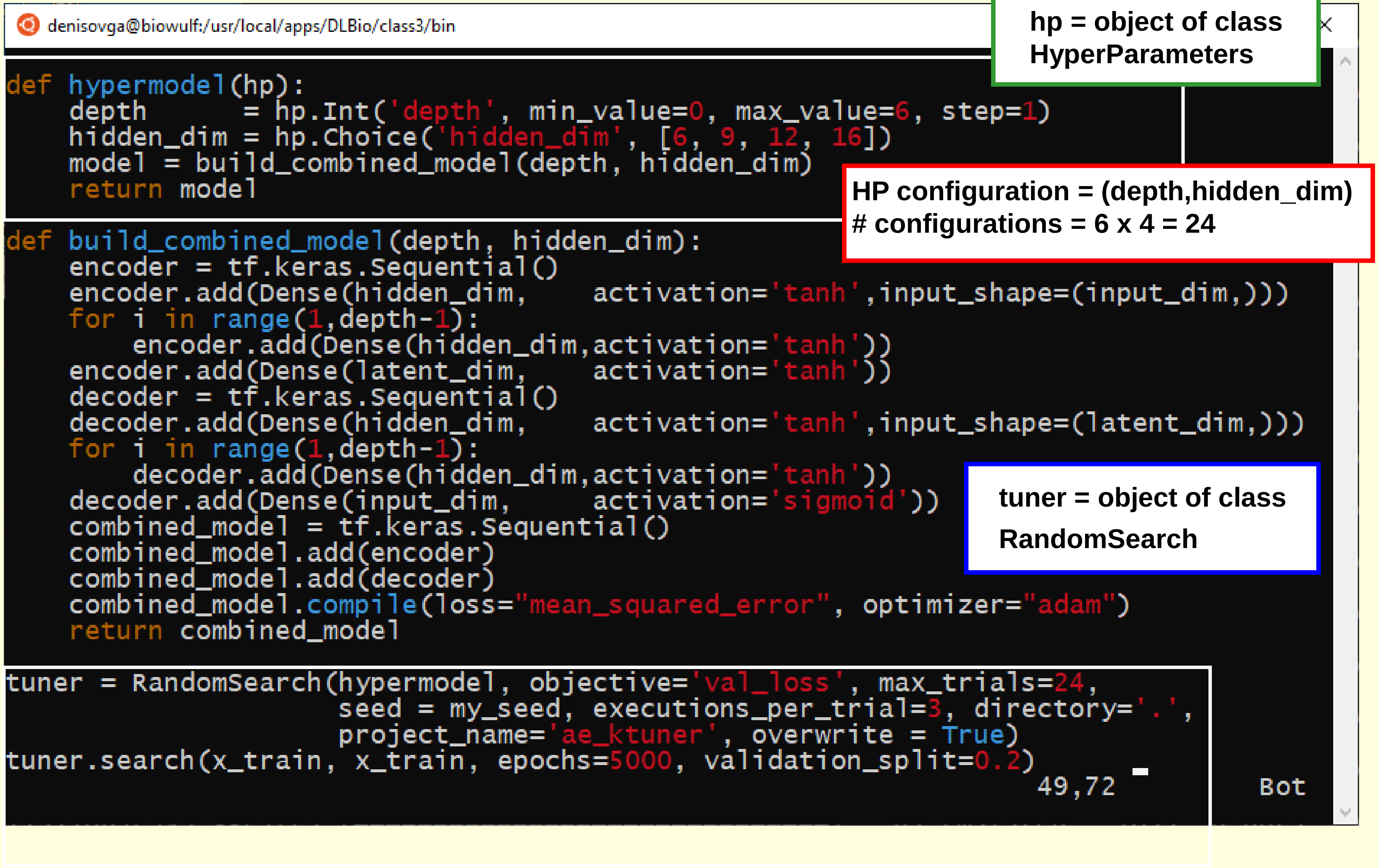
```
n_genes, n_samples, my_seed = 3,1000, 1
np.random.seed(my_seed)
tf.compat.v1.set_random_seed(my_seed)
data,prob = [],np.random.uniform(0,1,(n_samples,n_genes))
for i in range(n_samples):
    samples = []
    for j in range(n_genes):
        s_j = np.random.choice([0,1],1,p=[prob[i][j],1.-prob[i][j]])
        samples.append(s_j)
        for k in range(n_genes):
            s_k = np.random.choice([0,1],1,p=[prob[i][k],1.-prob[i][k]])
            if j < k: samples.append(s_j*s_k)
    data.append(samples)
x_train  = np.squeeze(np.array(data, dtype = float))
input_dim, latent_dim = x_train.shape[1], n_genes
                                                          21,55          Top
```



Hyperparameter optimization with KerasTuner (cont.)



hypermodel, tuner, search, HP configuration, objective, project_name, max_trials, executions_per_trial



denisovga@biowulf:/usr/local/apps/DLBio/class3/bin

hp = object of class HyperParameters

HP configuration = (depth,hidden_dim)
# configurations = 6 x 4 = 24

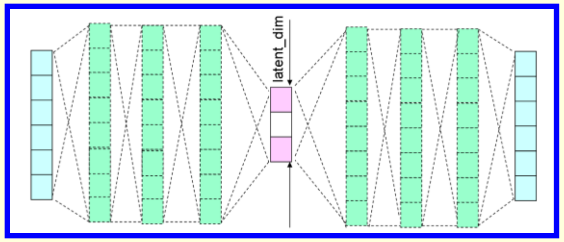
3) Define a model

```
def hypermodel(hp):
    depth      = hp.Int('depth', min_value=0, max_value=6, step=1)
    hidden_dim = hp.Choice('hidden_dim', [6, 9, 12, 16])
    model = build_combined_model(depth, hidden_dim)
    return model

def build_combined_model(depth, hidden_dim):
    encoder = tf.keras.Sequential()
    encoder.add(Dense(hidden_dim,    activation='tanh',input_shape=(input_dim,)))
    for i in range(1,depth-1):
        encoder.add(Dense(hidden_dim,activation='tanh'))
    encoder.add(Dense(latent_dim,    activation='tanh'))
    decoder = tf.keras.Sequential()
    decoder.add(Dense(hidden_dim,    activation='tanh',input_shape=(latent_dim,)))
    for i in range(1,depth-1):
        decoder.add(Dense(hidden_dim,activation='tanh'))
    decoder.add(Dense(input_dim,     activation='sigmoid'))
    combined_model = tf.keras.Sequential()
    combined_model.add(encoder)
    combined_model.add(decoder)
    combined_model.compile(loss="mean_squared_error", optimizer="adam")
    return combined_model
```

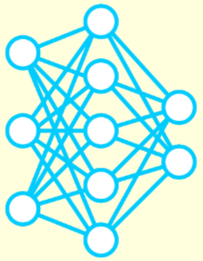
tuner = object of class RandomSearch

4) Run the model

```
tuner = RandomSearch(hypermodel, objective='val_loss', max_trials=24,
                     seed = my_seed, executions_per_trial=3, directory='.',
                     project_name='ae_ktuner', overwrite = True)
tuner.search(x_train, x_train, epochs=5000, validation_split=0.2)
                                                          49,72          Bot
```



Optimizing the latent dimension

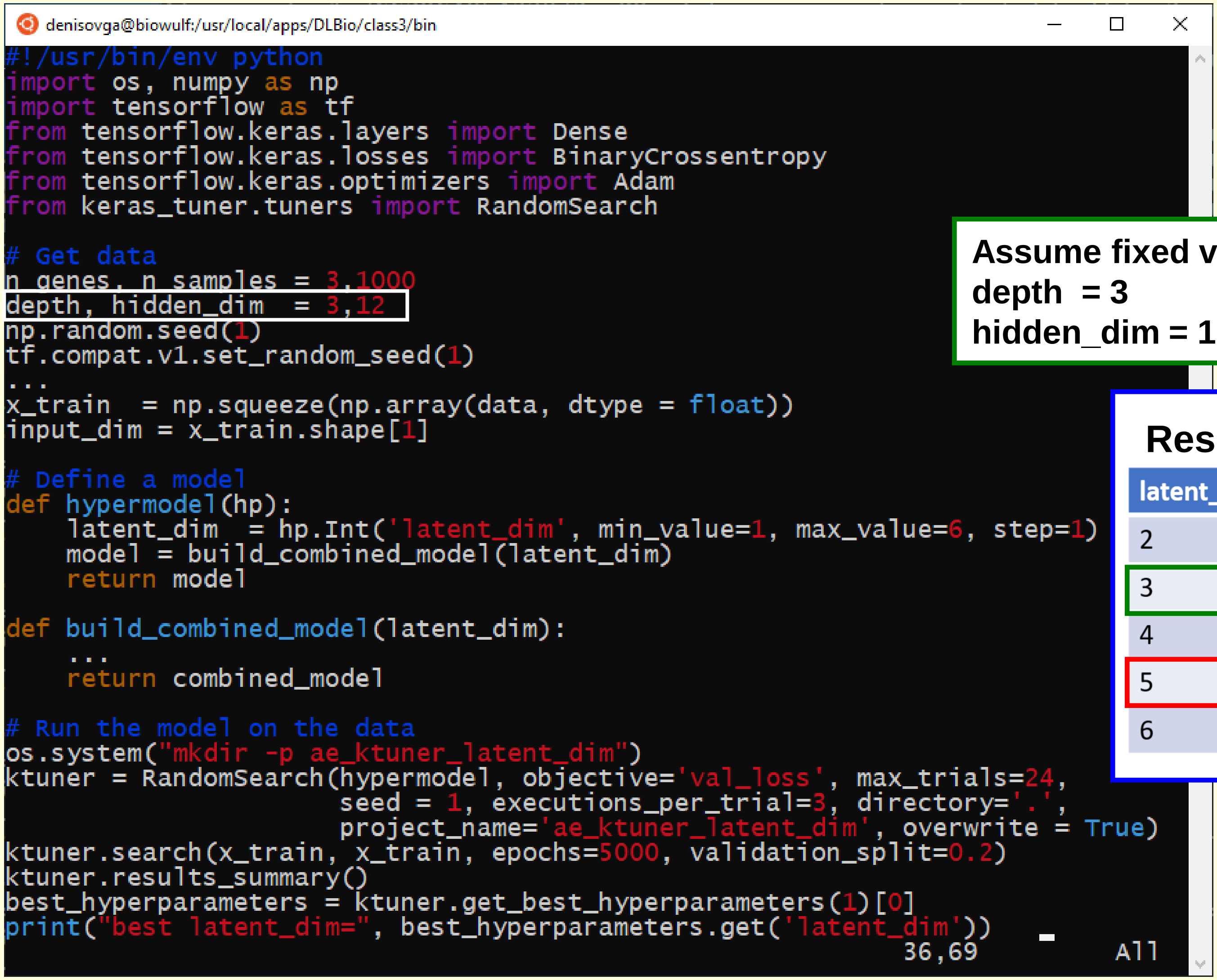


1) Header

2) Get data

3) Define a model

4) Run the model



denisovga@biowulf:/usr/local/apps/DLBio/class3/bin

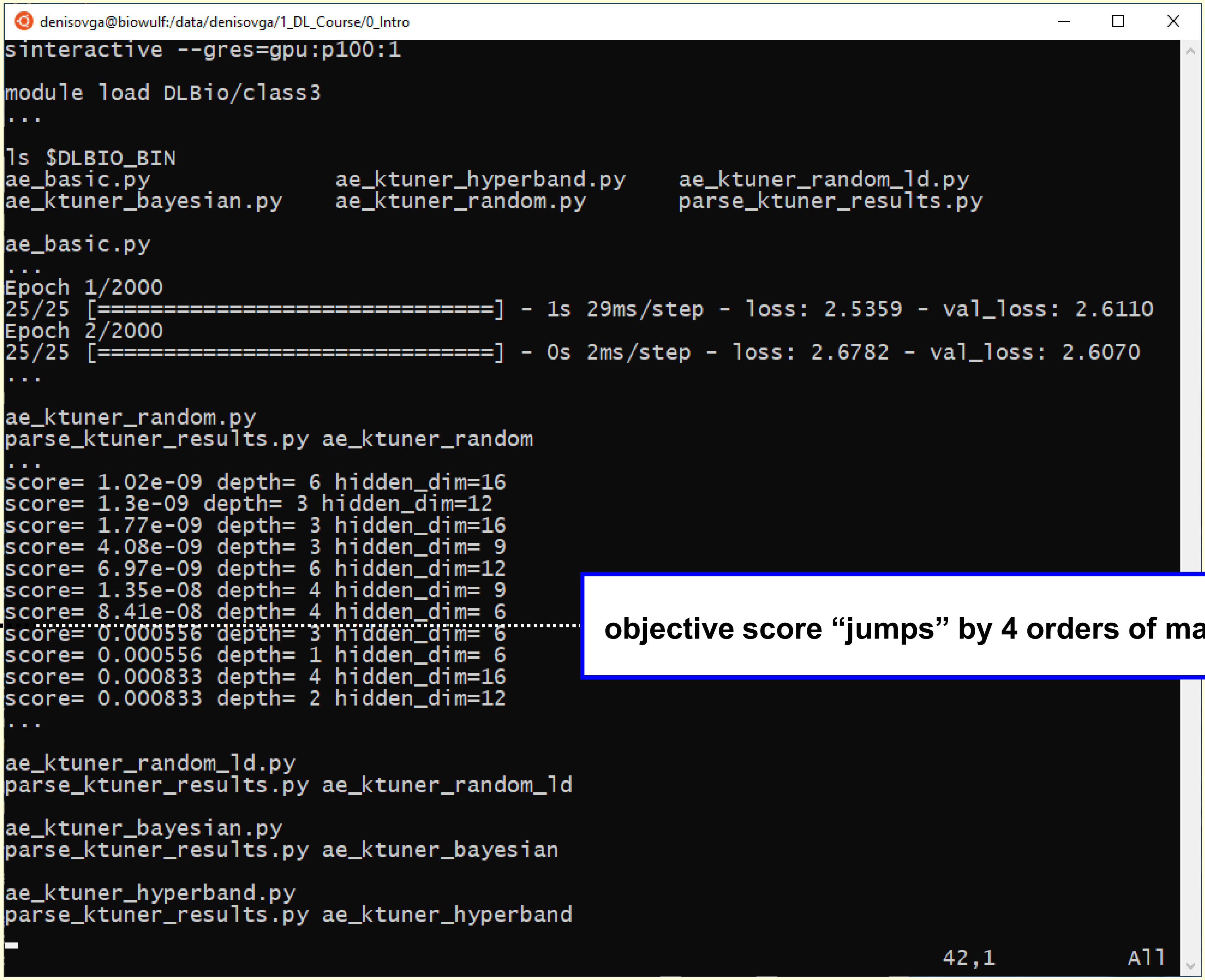
```
#!/usr/bin/env python
import os, numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.optimizers import Adam
from keras_tuner.tuners import RandomSearch

# Get data
n_genes, n_samples = 3,1000
depth, hidden_dim  = 3,12
np.random.seed(1)
tf.compat.v1.set_random_seed(1)
...
x_train  = np.squeeze(np.array(data, dtype = float))
input_dim = x_train.shape[1]

# Define a model
def hypermodel(hp):
    latent_dim  = hp.Int('latent_dim', min_value=1, max_value=6, step=1)
    model = build_combined_model(latent_dim)
    return model

def build_combined_model(latent_dim):
    ...
    return combined_model

# Run the model on the data
os.system("mkdir -p ae_ktuner_latent_dim")
ktuner = RandomSearch(hypermodel, objective='val_loss', max_trials=24,
                      seed = 1, executions_per_trial=3, directory='.',
                      project_name='ae_ktuner_latent_dim', overwrite = True)
ktuner.search(x_train, x_train, epochs=5000, validation_split=0.2)
ktuner.results_summary()
best_hyperparameters = ktuner.get_best_hyperparameters(1)[0]
print("best latent_dim=", best_hyperparameters.get('latent_dim'))
                                                            36,69          All
```

Assume fixed values:
depth = 3
hidden_dim = 12

Results:

latent_dim	score
2	0.0008
3	3.56e-7
4	1.41e-7
5	1.21e-7
6	1.28e-7

How to run the simple/prototype models on Biowulf?



denisovga@biowulf:/data/denisovga/1_DL_Course/0_Intro

```
sinteractive --gres=gpu:p100:1

module load DLBio/class3
...

ls $DLBIO_BIN
ae_basic.py              ae_ktuner_hyperband.py    ae_ktuner_random_ld.py
ae_ktuner_bayesian.py    ae_ktuner_random.py       parse_ktuner_results.py

ae_basic.py
...
Epoch 1/2000
25/25 [==============================] - 1s 29ms/step - loss: 2.5359 - val_loss: 2.6110
Epoch 2/2000
25/25 [==============================] - 0s 2ms/step - loss: 2.6782 - val_loss: 2.6070
...

ae_ktuner_random.py
parse_ktuner_results.py ae_ktuner_random
...
score= 1.02e-09 depth= 6 hidden_dim=16
score= 1.3e-09 depth= 3 hidden_dim=12
score= 1.77e-09 depth= 3 hidden_dim=16
score= 4.08e-09 depth= 3 hidden_dim= 9
score= 6.97e-09 depth= 6 hidden_dim=12
score= 1.35e-08 depth= 4 hidden_dim= 9
score= 8.41e-08 depth= 4 hidden_dim= 6
score= 0.000556 depth= 3 hidden_dim= 6
score= 0.000556 depth= 1 hidden_dim= 6
score= 0.000833 depth= 4 hidden_dim=16
score= 0.000833 depth= 2 hidden_dim=12
...

ae_ktuner_random_ld.py
parse_ktuner_results.py ae_ktuner_random_ld

ae_ktuner_bayesian.py
parse_ktuner_results.py ae_ktuner_bayesian

ae_ktuner_hyperband.py
parse_ktuner_results.py ae_ktuner_hyperband

                                                            42,1          All
```

objective score “jumps” by 4 orders of magnitude

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

Tybalt paper: J.P.Way, C.S.Greene, Pacif Symp. on Biocomputing (2018)
Tybalt orig.code: https://github.com/greenelab/tybalt
Tybalt on Biowulf: https://hpc.nih.gov/apps/Tybalt.html

Input: 20,530 gene expression profiles in
10,459 samples representing 33 types of cancer**:**
- 9,732 tumor samples
- 727 normal samples

Task: Extract a biologically relevant latent space from the transcriptome

Data from: **TCGA**
(The Cancer Genome Altas)
- NIH program led by NCI and NHGRI

Steps:

1) **Preprocessing:** extract a subset of genes with the most variable expression profiles (**20,530 → 5,000**)

2) **Production** (involves **deep learning**)**:**
reduce the dimensionality of the feature space by 50 fold (**5000 → 100**) using
variational autoencoder.
For comparison, the same task will also be performed by
denoising autoencoder

3) **Postprocessing:** verify that samples encoded by autoencoder **retain biological signals**

Overview of the Tybalt training code

(only the main function is shown)

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

↑

Imports statements, other function definitions

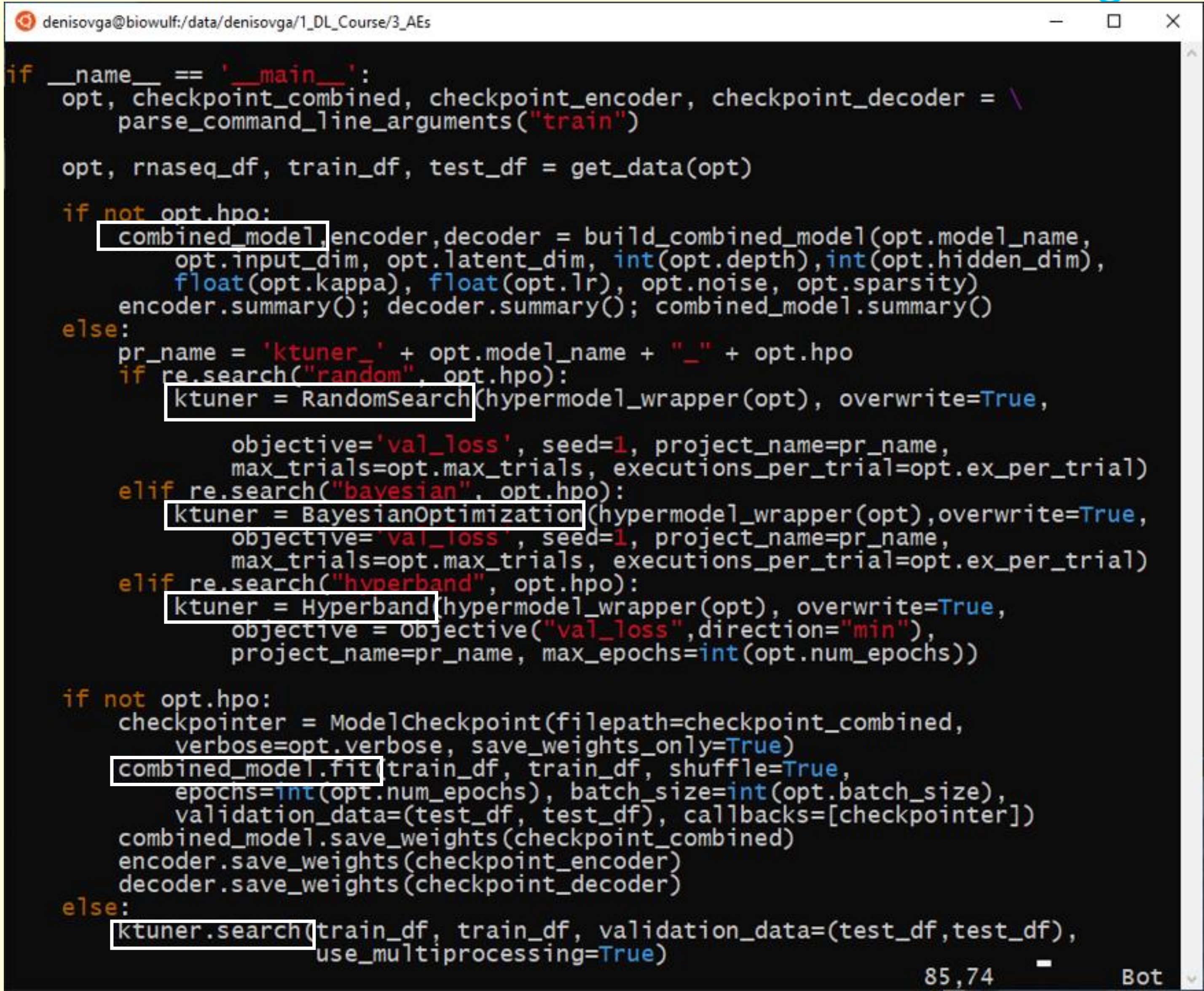
Header
- parse the command line options

Getting data
- data in TSV format

Defining a model
- models: VAE, ADAGE
- tuners:
  RandomSearch
  BayesianOptimization
  Hyperband

Running the model
- fit
- search

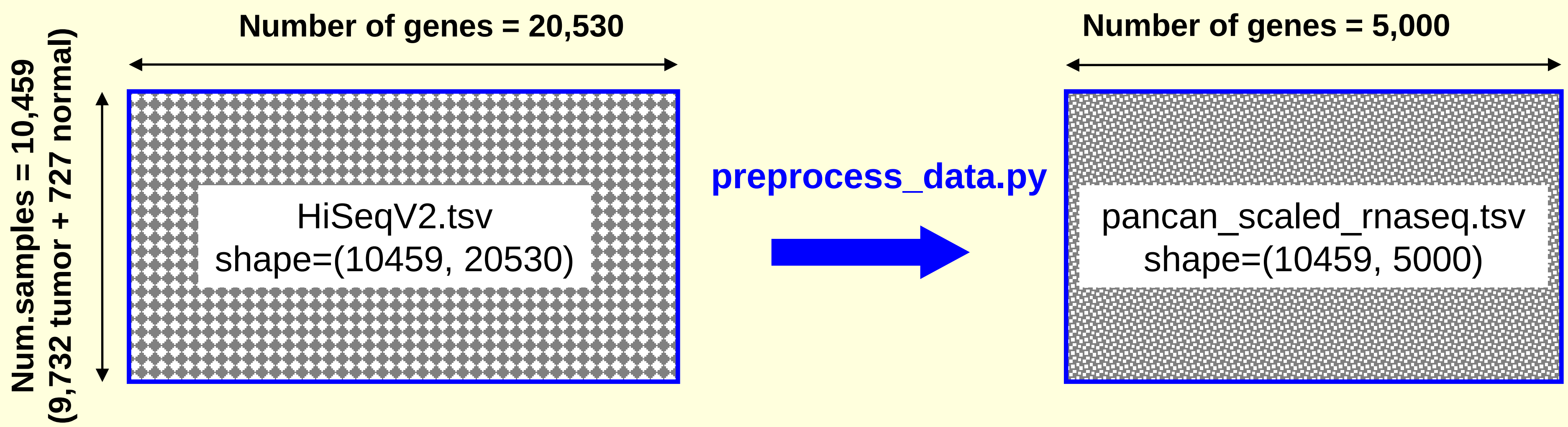
Extra:
- tSNE



```
if __name__ == '__main__':
  opt, checkpoint_combined, checkpoint_encoder, checkpoint_decoder = \
    parse_command_line_arguments("train")

  opt, rnaseq_df, train_df, test_df = get_data(opt)

  if not opt.hpo:
    combined_model,encoder,decoder = build_combined_model(opt.model_name,
      opt.input_dim, opt.latent_dim, int(opt.depth),int(opt.hidden_dim),
      float(opt.kappa), float(opt.lr), opt.noise, opt.sparsity)
    encoder.summary(); decoder.summary(); combined_model.summary()
  else:
    pr_name = 'ktuner_' + opt.model_name + "_" + opt.hpo
    if re.search("random", opt.hpo):
      ktuner = RandomSearch(hypermodel_wrapper(opt), overwrite=True,

        objective='val_loss', seed=1, project_name=pr_name,
        max_trials=opt.max_trials, executions_per_trial=opt.ex_per_trial)
    elif re.search("bayesian", opt.hpo):
      ktuner = BayesianOptimization(hypermodel_wrapper(opt),overwrite=True,
        objective='val_loss', seed=1, project_name=pr_name,
        max_trials=opt.max_trials, executions_per_trial=opt.ex_per_trial)
    elif re.search("hyperband", opt.hpo):
      ktuner = Hyperband(hypermodel_wrapper(opt), overwrite=True,
        objective = Objective("val_loss",direction="min"),
        project_name=pr_name, max_epochs=int(opt.num_epochs))

  if not opt.hpo:
    checkpointer = ModelCheckpoint(filepath=checkpoint_combined,
      verbose=opt.verbose, save_weights_only=True)
    combined_model.fit(train_df, train_df, shuffle=True,
      epochs=int(opt.num_epochs), batch_size=int(opt.batch_size),
      validation_data=(test_df, test_df), callbacks=[checkpointer])
    combined_model.save_weights(checkpoint_combined)
    encoder.save_weights(checkpoint_encoder)
    decoder.save_weights(checkpoint_decoder)
  else:
    ktuner.search(train_df, train_df, validation_data=(test_df,test_df),
           use_multiprocessing=True)
                                               85,74          Bot
```

Tybalt data

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

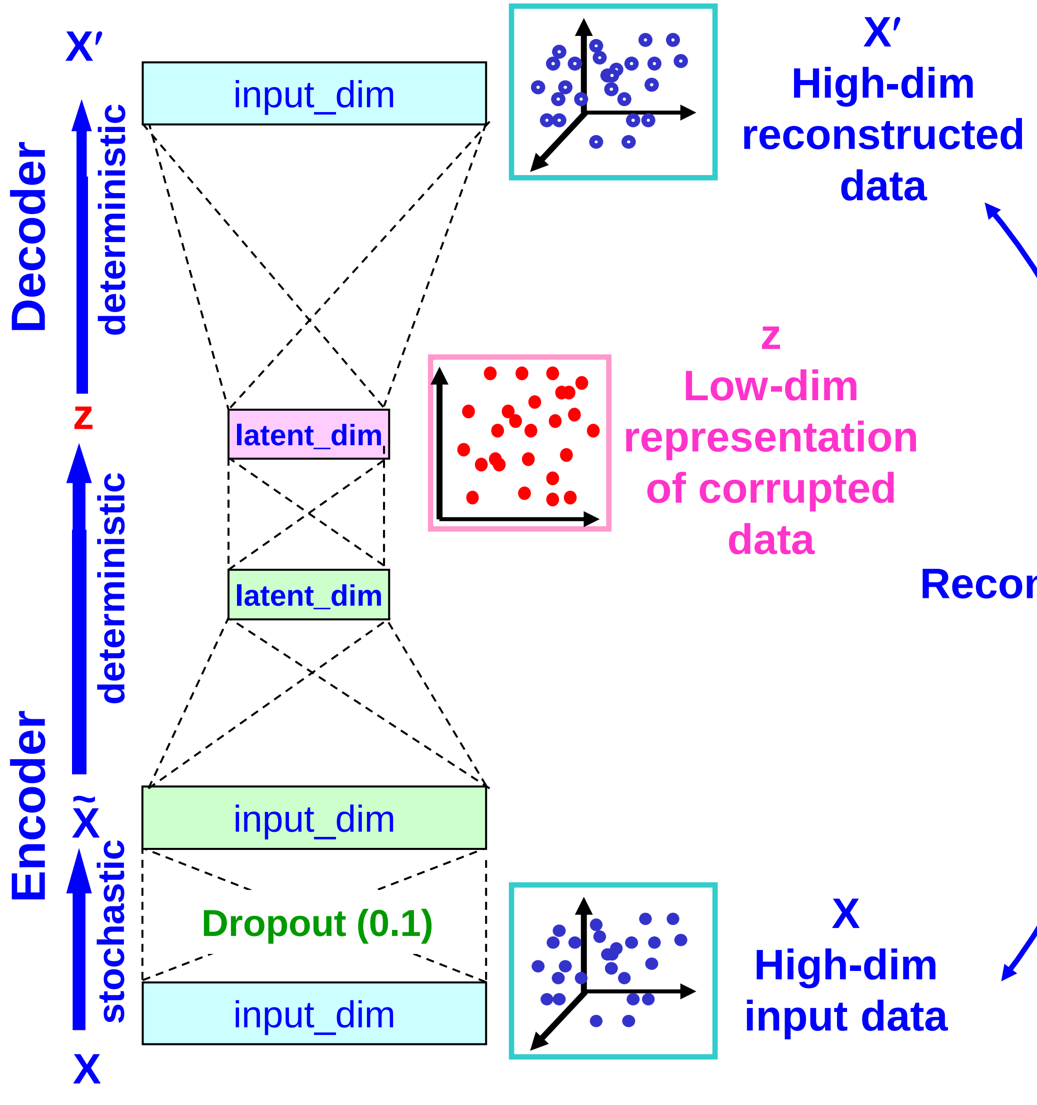


Other raw data	Shape
Gistic2_CopyNumber_all_thresholded.by_genes	(24776, 10845)
PANCAN_mutation	(2034801, 10)
samples.tsv	(11284, 860)
PANCAN_clinicalMatrix	(12088, 35)

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

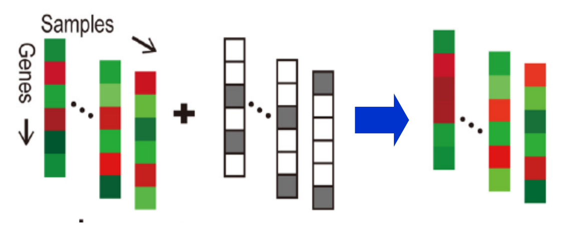
ADAGE paper: J.Tan et al., mSystems (2016)



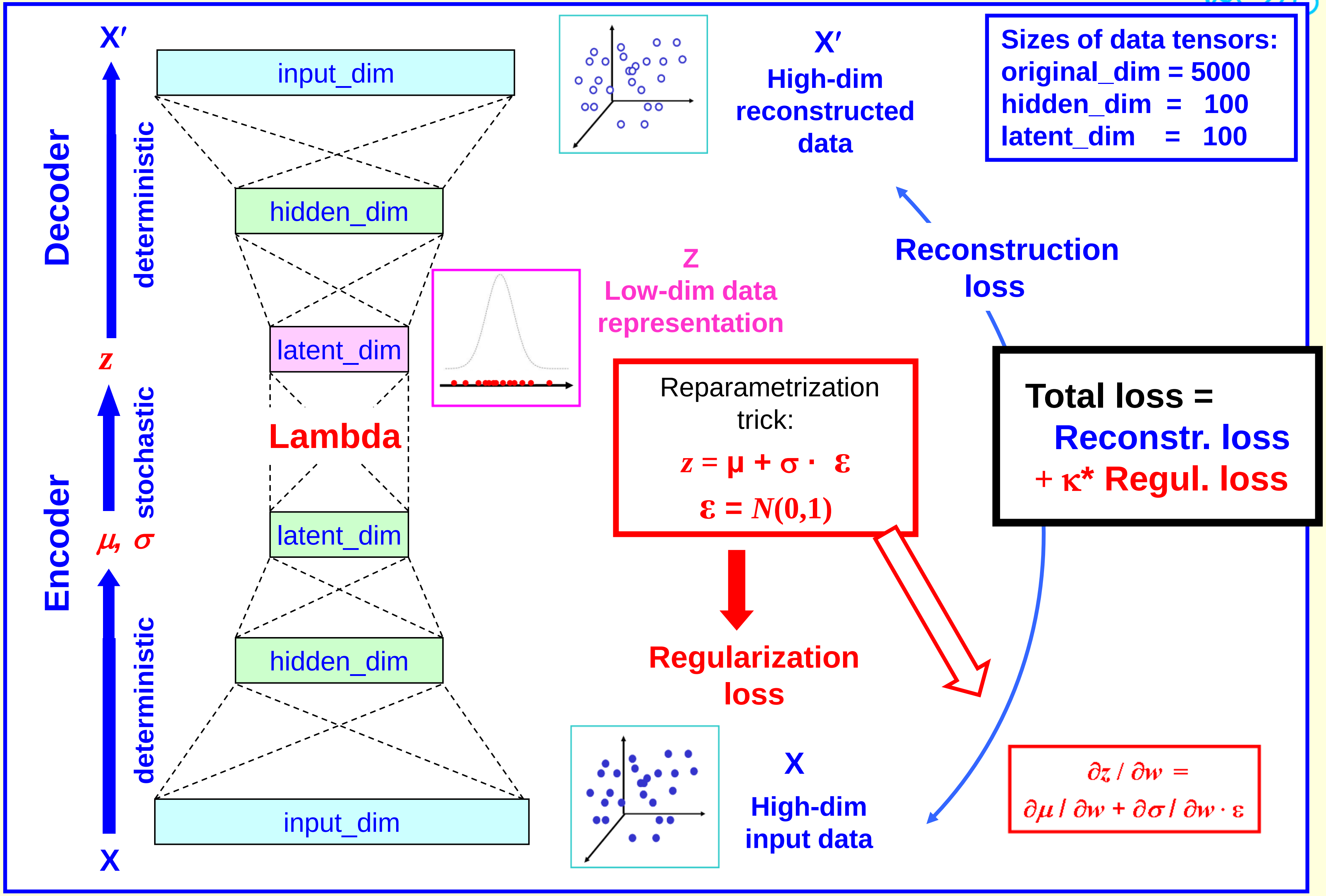
Reconstruction loss $= ||X - X'||$

$\Leftrightarrow$ MSE(X, X′) $\rightarrow$ min

Sizes of data tensors:
- original_dim = 5,000
- latent_dim = 100



The VAE (variational autoencoder) model



Sizes of data tensors:
original_dim = 5000
hidden_dim = 100
latent_dim = 100

Reparametrization trick:

$$z = \mu + \sigma \cdot \varepsilon$$

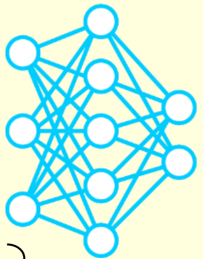
$$\varepsilon = N(0,1)$$

Total loss =
Reconstr. loss
+ κ* Regul. loss

$$\partial z / \partial w = \partial \mu / \partial w + \partial \sigma / \partial w \cdot \varepsilon$$

Hyperparameter optimization with KerasTuner

CANDLE (Grid, Bayesian; parallel): https://hpc.nih.gov/apps/candle/index.html

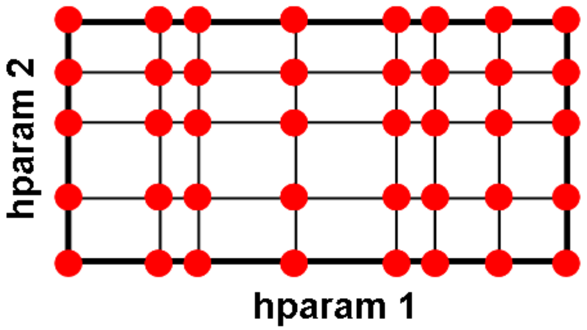


Tunable HP	depth	hidden_dim	κ	batch_size	num_epochs	learning_rate
Default range of var. (from orig. paper)	[0, 1]	[100, 300]	[0.01, 0.05, 0.1, 1.]	[50, 100, 128, 200]	[10, 25, 50, 100]	[0.0005, 0.001, 0.0015, 0.002, 0.0025]

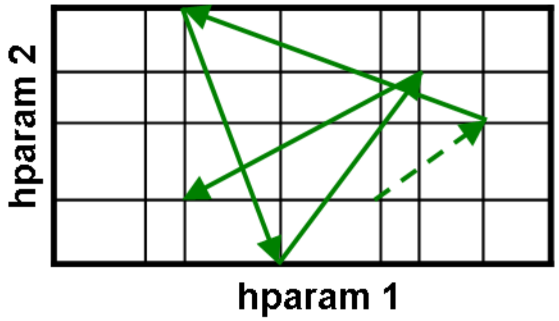
1280 HP configs total

Keras tuners: RandomSearch, BayesianOptimization, Hyperband, Sklearn

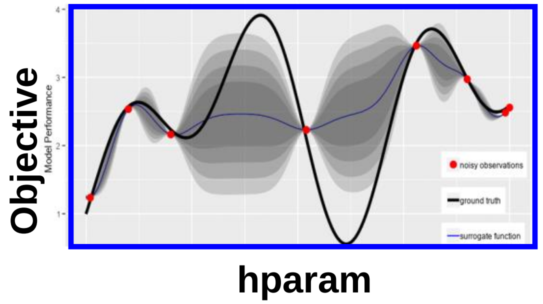
Grid (CANDLE)



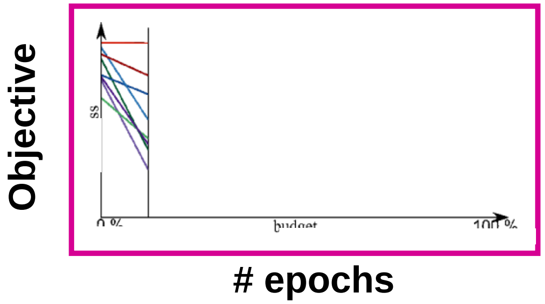
RandomSearch



BayesianOptimization

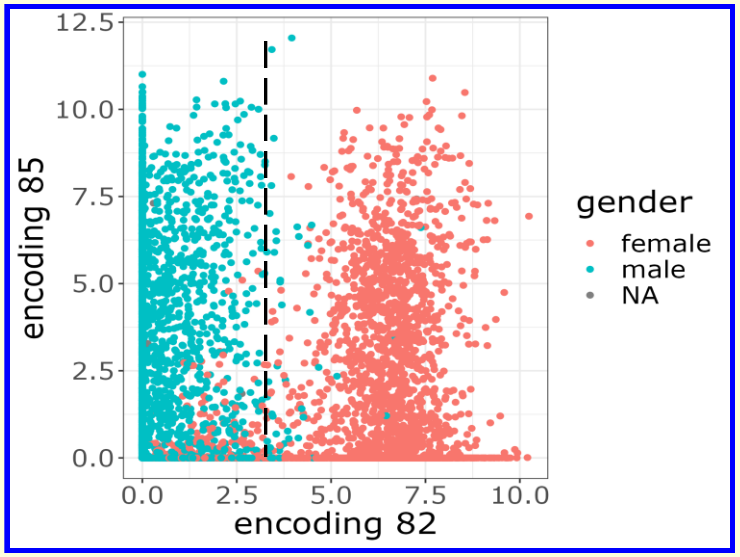


Hyperband

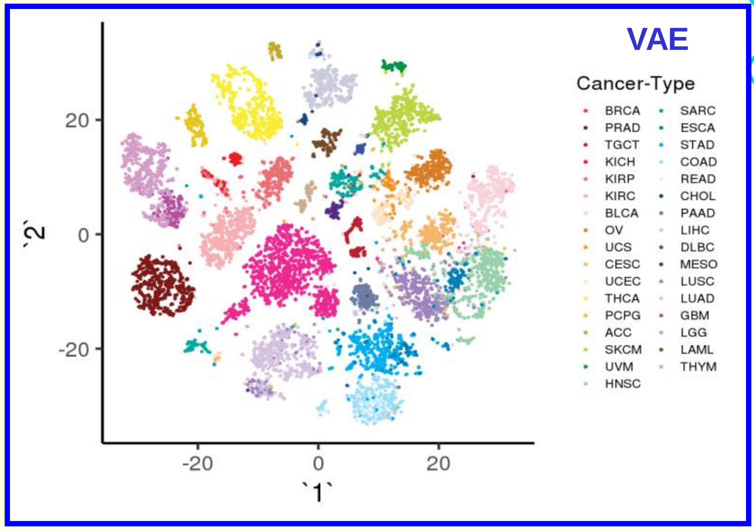


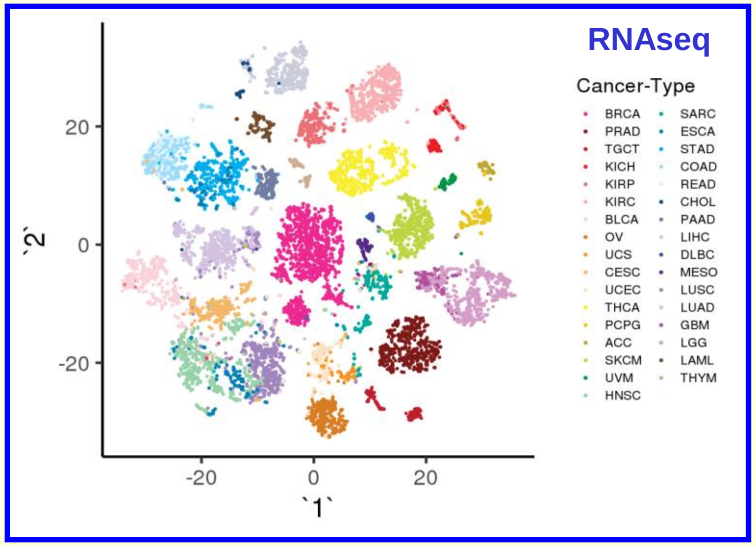
Sklearn

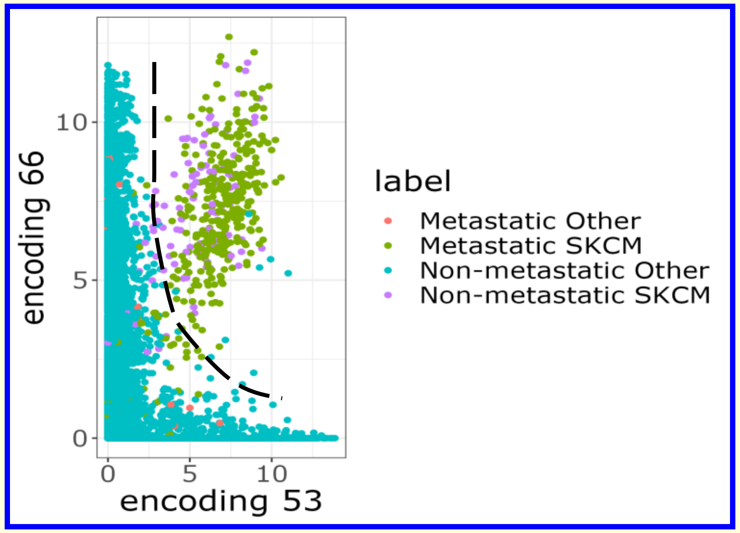
Samples encoded by VAE retain biological signals



Encoding 82 stratifies patient sex







Encodings 53 and 66 separate melanoma tumors

tSNE of VAE-encoded samples (100$\rightarrow$ 2) preserve the same clusters as tSNE of unencoded RNAseq samples (5000 $\rightarrow$ 2).

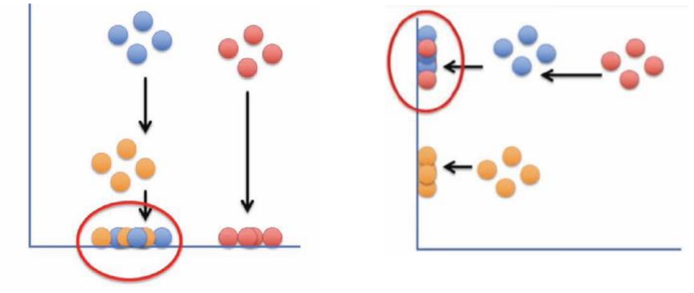
tSNE: t-distributed Stochastic Neighbor Embedding

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

Application to SC transcriptomics: D.Koback, P.Berens - Nature Comm. (2019) 10:5416

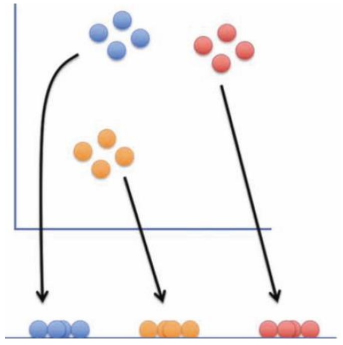
Task:
map data points,
<u>together with their neighbors,</u>
from a high-dim "input" space
(e.g. dim=100 or 5000)
to a low-dim "embedding"
space (dim=2),
for subsequent visualization

Projections do not preserve the structure of clusters



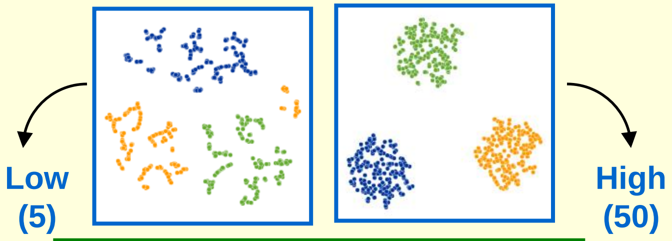
to x-axis **to y-axis**

tSNE



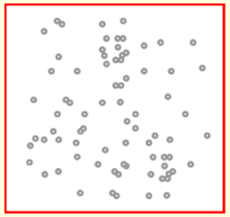
```
sklearn.manifold.TSNE(n_components=2,
                      perplexity=30.0,
                      init='random',
                      learning_rate=200.0, … )
```

<u>Perplexity</u>: effective # neighbors of a data point;



Tybalt's choice: Perp = 20

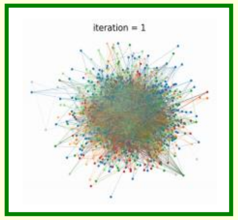
Initialization: starting data distribution in the low-dim space



Random

PCA

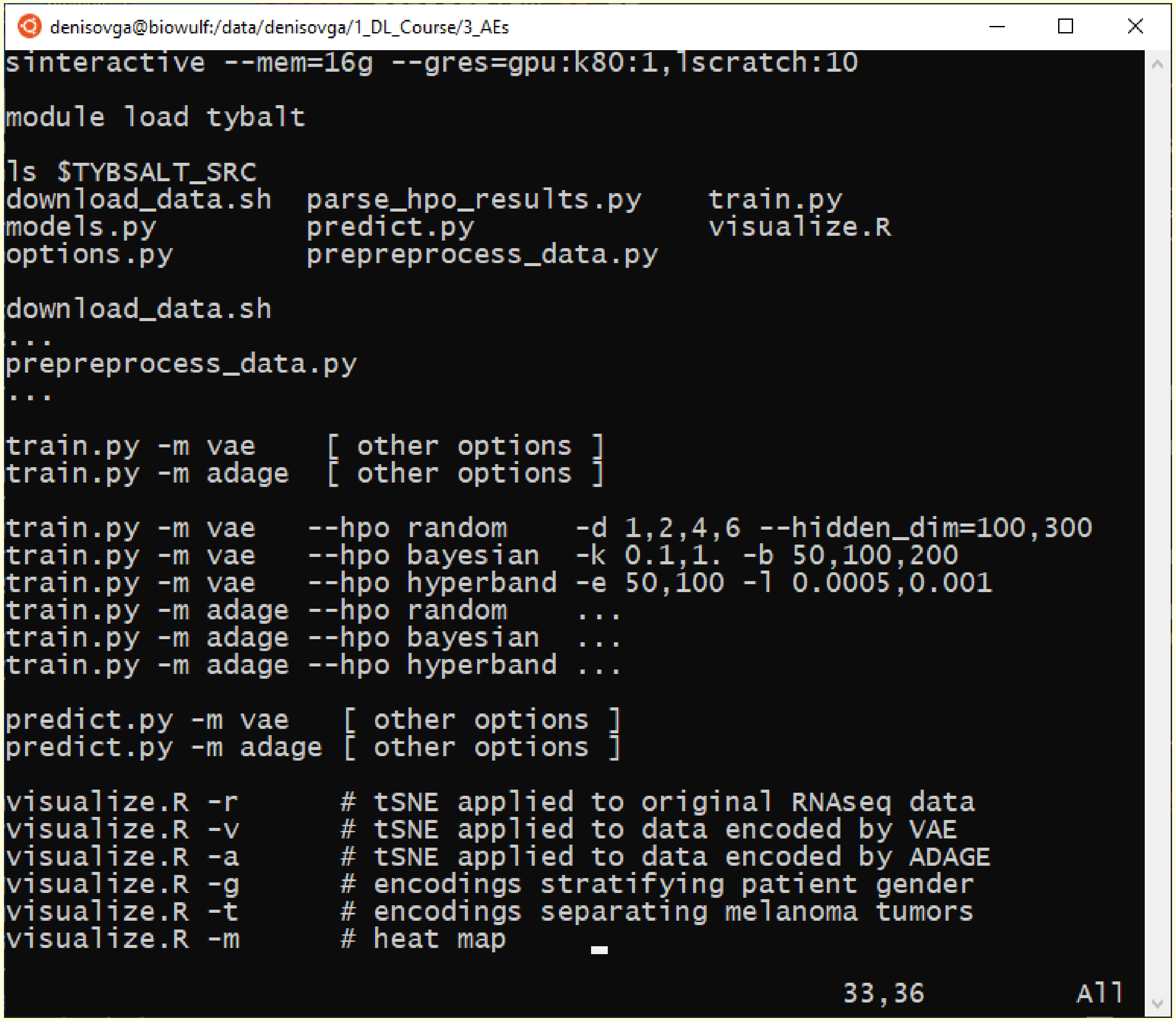
Learning rate:
N = 10,459 data size



η = max(200, N/12)

How to run the Tybalt application on Biowulf?

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



denisovga@biowulf:/data/denisovga/1_DL_Course/3_AEs

```
sinteractive --mem=16g --gres=gpu:k80:1,lscratch:10

module load tybalt

ls $TYBSALT_SRC
download_data.sh  parse_hpo_results.py    train.py
models.py         predict.py              visualize.R
options.py        prepreprocess_data.py

download_data.sh
...
prepreprocess_data.py
...

train.py -m vae     [ other options ]
train.py -m adage   [ other options ]

train.py -m vae   --hpo random    -d 1,2,4,6 --hidden_dim=100,300
train.py -m vae   --hpo bayesian  -k 0.1,1. -b 50,100,200
train.py -m vae   --hpo hyperband -e 50,100 -l 0.0005,0.001
train.py -m adage --hpo random    ...
train.py -m adage --hpo bayesian  ...
train.py -m adage --hpo hyperband ...

predict.py -m vae    [ other options ]
predict.py -m adage [ other options ]

visualize.R -r      # tSNE applied to original RNAseq data
visualize.R -v      # tSNE applied to data encoded by VAE
visualize.R -a      # tSNE applied to data encoded by ADAGE
visualize.R -g      # encodings stratifying patient gender
visualize.R -t      # encodings separating melanoma tumors
visualize.R -m      # heat map

                                                   33,36         All
```

Summary

1) Intro using a simple example
- basic shallow and deep autoencoders (AEs): the shallow AE 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
- data with larger number of components require a deeper AE model with larger intermediate data tensors

2) Hyperparameter optimization with KerasTuner
- the task of optimizing latent dimension can be formulated as HPO problem
- hypermodel, hyperparameter configuration, trial, executions per trial
- the tuner object and the search method

3) The biological example
- ADAGE (denoising autoencoder) model
- VAE (variational autoencoder) model, reparametrization trick, reconstruction and regularization losses
- Grid search
- Keras tuners: RandomSearch, BayesianOptimization, Hyperband
- the VAE encodings retain biological signals
- tSNE for visualization of high-dimensional data