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
  - 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

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

**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

<table>
<thead>
<tr>
<th>Samples</th>
<th>X</th>
<th>Y</th>
<th>XY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Shallow 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 model**

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

1) Header

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

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))

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

2) Get data

3) Define a model

4) Run the model
# Results for the basic models: deep autoencoder vs PCA


<table>
<thead>
<tr>
<th>epoch #</th>
<th>BCE val_loss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Shallow model</td>
</tr>
<tr>
<td>200</td>
<td>2.5194</td>
</tr>
<tr>
<td>400</td>
<td>2.5194</td>
</tr>
<tr>
<td>600</td>
<td>2.5194</td>
</tr>
<tr>
<td>800</td>
<td>2.5194</td>
</tr>
<tr>
<td>1000</td>
<td>2.5194</td>
</tr>
<tr>
<td>1200</td>
<td>2.5194</td>
</tr>
<tr>
<td>1400</td>
<td>2.5194</td>
</tr>
<tr>
<td>1600</td>
<td>2.5194</td>
</tr>
<tr>
<td>1800</td>
<td>2.5194</td>
</tr>
<tr>
<td>2000</td>
<td>2.5194</td>
</tr>
</tbody>
</table>

**Conclusions:**

1) The shallow model with linear activation, which mimics the PCA, 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/

1) Header

```python
#!/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

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
```

2) Get data

```
#denisovga@bioculus:/data/denisovga/1_DL_Course/0_Intro
```

Model

Data matrix

```
X       Y       Z       XY      XZ      YZ
   R       B       G   R*G   R*B   G*B
   B       G       R   B*G   B*R   G*R
   G       R       B   G*R   G*B   R*B
```

depth

hidden_dim

...
Hyperparameter optimization with KerasTuner (cont.)

3) Define a model

```python
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
```

4) Run the model

```python
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
```

```python
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)
```

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

hp = object of class HyperParameters
tuner = object of class RandomSearch
Optimizing the latent dimension

1) Header

```python
#!/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]
```

Assume fixed values:

- depth = 3
- hidden_dim = 12

2) Get data

```python
# 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
```

3) Define a model

```python
# 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'))
```

4) Run the model

Results:

<table>
<thead>
<tr>
<th>latent_dim</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0008</td>
</tr>
<tr>
<td>3</td>
<td>3.56e-7</td>
</tr>
<tr>
<td>4</td>
<td>1.41e-7</td>
</tr>
<tr>
<td>5</td>
<td>1.21e-7</td>
</tr>
<tr>
<td>6</td>
<td>1.28e-7</td>
</tr>
</tbody>
</table>
How to run the simple/prototype models on Biowulf?

```
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
Epoch 2/2000
...
```

objective score “jumps” by 4 orders of magnitude
Example 3. Tybalt: extracting a biologically relevant latent space from cancer transcriptomes

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

**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.

Data from: TCGA (The Cancer Genome Atlas)
- NIH program led by NCI and NHGRI
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
Tybalt data

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

Raw RNA-seq gene expression data (downloaded)

- Num. samples = 10,459 (9,732 tumor + 727 normal)
- Number of genes = 20,530
- HiSeqV2.tsv
  shape=(10459, 20530)

Preprocessed RNA-seq gene expression data (used as input by the DL code)

- Number of genes = 5,000
- preprocess_data.py
  pancy_scaled_rnaseq.tsv
  shape=(10459, 5000)

Other raw data

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

Other processed data

- pancy_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)

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

Reconstruction loss = ||X - X'|| = MSE(X, X') → min

Encoder:
- stochastic
- Dropout (0.1)
- input_dim
- latent_dim

Decoder:
- deterministic
- input_dim
- latent_dim
- input_dim
- X'

X

High-dim input data

X'

High-dim reconstructed data

z

Low-dim representation of corrupted data

Genes

Samples

+ -->

Output

High-dim reconstructed data
The VAE (variational autoencoder) model

Encoder

- \( \mu, \sigma \)
  - deterministic

- \( \text{latent\_dim} \)
  - stochastic

Decoder

- \( \text{input\_dim} \)
- \( \text{hidden\_dim} \)
- \( \text{latent\_dim} \)

Reparametrization trick:

\[
\mathbf{z} = \mu + \sigma \cdot \mathbf{\xi}
\]

\( \mathbf{\xi} \sim \mathcal{N}(0,1) \)

Sizes of data tensors:
- original\_dim = 5000
- hidden\_dim = 100
- latent\_dim = 100

Reconstruction loss

Total loss = Reconstr. loss + \( \kappa \) * Regul. loss

Explicit regularization loss:

\[
\frac{\partial \mathbf{z}}{\partial \mathbf{w}} = \frac{\partial \mu}{\partial \mathbf{w}} + \frac{\partial \sigma}{\partial \mathbf{w}} \cdot \mathbf{\xi}
\]
**Hyperparameter optimization**

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

<table>
<thead>
<tr>
<th>Tunable HP</th>
<th>depth</th>
<th>hidden_dim</th>
<th>$\kappa$</th>
<th>batch_size</th>
<th>num_epochs</th>
<th>learning_rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range of var.</td>
<td>[0, 1]</td>
<td>[100, 300]</td>
<td>[0.01, 0.05, 0.1, 1]</td>
<td>[50, 100, 128, 200]</td>
<td>[10, 25, 50, 100]</td>
<td>[0.0005, 0.001, 0.0015, 0.002, 0.0025]</td>
</tr>
</tbody>
</table>

**Keras tuners:** 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 → 2) preserve the same clusters as tSNE of unencoded RNAseq samples (5000 → 2).
tSNE: t-distributed Stochastic Neighbor Embedding

**Task:**
map data points, together with their neighbors, 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

![Projections](image)

**tSNE**

```
from sklearn.manifold import TSNE

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

**Perplexity:** effective # neighbors of a data point;

Low (5)  | High (50)
Tybalt’s choice: **Perp = 20**

**Initialization:** starting data distribution in the low-dim space

- Random
- PCA

**Learning rate:**

\[ \eta = \max(200, N/12) \]


How to run the Tybalt application on Biowulf?

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