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
  - Tybalt: classification of cancer samples based on gene expression
Examples overview

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

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)
Input
A layer of 2 neurons
Output

Model 2 (deep)
Input
Hidden layers
Output

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)
- activations: linear (Model 1) or sigmoid (Model 2)
- size of the latent space $Z$: 2
- sizes of other tensors ($X, Y: 9; \ X_1, Z_1: 5$)

Deep network: $\geq 2$ 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
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
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?

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


[https://hpc.nih.gov/apps/Tybalt.html](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

---

**Steps:**

1. **download_data.sh**
2. **Raw data (TSV)**
   - Pre-processed data (TSV)
   - process_data.py
   - Checkpoints (HDF5)
     - Encoded data; tSNE features (TSV)
     - Plots (PNG)

**Scripts:**
- tybalt_train.py
- tybalt_predict.py
- tybalt_visualize.R

---

**Deep Learning**
Overview of the Tybalt training code

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

Header
- parse the command line options

---

```python
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)
    checkpoint = 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), checkpoint])
    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=[checkpoint])
    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

Raw data
(downloadned)

Number of genes = 20,530

HiSeqV2
(TSV)

(Pre-)processed data
(used as input by the DL code)

Num. samples = 10,459
(9,732 tumor + 727 normal)

Number of genes = 5,000

process_data.py

Other raw data

<table>
<thead>
<tr>
<th>Shape</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gistic2_CopyNumber_all_thresholded.by_genes</td>
<td><a href="https://zenodo.org/record/827323#.XPexAFVKhhE">https://zenodo.org/record/827323#.XPexAFVKhhE</a></td>
</tr>
<tr>
<td>PANCAN_mutation</td>
<td><a href="http://snaptron.cs.jhu.edu/data/tcga/samples.tsv">http://snaptron.cs.jhu.edu/data/tcga/samples.tsv</a></td>
</tr>
<tr>
<td>samples.tsv</td>
<td>(2034801, 10)</td>
</tr>
<tr>
<td>PANCAN_clinicalMatrix</td>
<td>(12088, 35)</td>
</tr>
</tbody>
</table>

Other processed data

<table>
<thead>
<tr>
<th>Shape</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>pancer_mutation.tsv</td>
<td>(7515, 29829)</td>
</tr>
<tr>
<td>status_matrix.tsv</td>
<td>(7230, 29829)</td>
</tr>
<tr>
<td>tybalt_features_with_clinical.tsv</td>
<td>10375, 117</td>
</tr>
</tbody>
</table>
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

Dropout (0.1)
The VAE (variational autoencoder) model

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

Reparametrization trick:
\[ z = \mu + \sigma \cdot \varepsilon \]
\[ \varepsilon = N(0,1) \]

\[ \frac{\partial z}{\partial w} = \frac{\partial \mu}{\partial w} + \frac{\partial \sigma}{\partial w} \cdot \varepsilon \]
tSNE: t-Distributed Stochastic Neighbor Embedding

**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)

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.

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

\[
Q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_k - y_i\|^2)^{-1}}
\]
How to run the Tybalt application on Biowulf?

https://hpc.nih.gov/apps/tybalt.html
https://github.com/greenelab/tybalt

![Image of shell commands]

```
$ sinteractive --mem=16g \
   --gres=gpu:p100:1,lscratch:10
$ module load tybalt
$ ls $TYBALT_SRC
$ download_data.sh
$ download_data.sh
$ process_data.py
$ tybalt_train.py -m vae
$ tybalt_predict.py -m vae
$ tybalt_visualize.R
$ tybalt_train.py -m adage
$ tybalt_predict.py -m adage
$ tybalt_visualize.R -a
```

![VAE model + tSNE graph]

![AGAGE model + tSNE graph]
More on the gradient descent-based optimizers: momentum and Nesterov accelerated gradient

\[ \Delta w_t = -\gamma \cdot \nabla_w J(w_t) \]

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

- small \( \gamma \rightarrow \text{slow convergence along the valley} \)
- larger \( \gamma \rightarrow \text{oscillations in the perpendicular dir.} \)

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

- gradient descent formula with momentum \( \mu \) (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 \( \mu \) and Nesterov accelerated gradient

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

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.

Deterministic approach

Objective: loss(X, X') → min

Probabilistic approach

Objective: P(X) → max

Deterministic approach:

Encoder

X / X'

Decoder

z

Probabilistic approach:

P(z|X)

P(X)

P(z)

log P(X) = log ∫ P(X|z) P(z) dz → max

log P(X) = E_{z~P(z|X)} P(X|z) - D_{KL}[P(X|z) \| P(z)] → max

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

"evidence" always >= 0

Reconstruction loss

Regularization loss

Keras implementation

loss = - ELBO ≈ BC(X, X') - \beta \cdot \sum [\sigma_i(X) + \mu_i(X)^2 - 1 - log \sigma_i(X)] → min

BC = Binary cross-entropy

D_{KL} = Kullback-Leibler divergence:

D_{KL}[p(x) \| q(x)] = ∫ p(x) \cdot ln[p(x)/q(x)]