Deep Learning by Example on Biowulf

3. Autoencoders

Gennady Denisov, PhD
Intro and goals

What is autoencoder?

Two basic requirements:
1) The input and output tensors have the same number of units
2) At least one of the intermediate data tensors has a smaller number of active units than the input and output tensors

Examples:

Denoising autoencoder

Variational (probabilistic) autoencoder

ADAGE: analysis using denoising autoencoders of gene expression

Generating images

Tybalt: clustering of cancer samples based on gene expression
Examples overview

<table>
<thead>
<tr>
<th>#</th>
<th>Biological Application</th>
<th>Network type</th>
<th>ML type</th>
<th>How #3 differs from #1 and #2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bioimage segmentation/ fly brain</td>
<td>Convolutional Neural</td>
<td>Supervised</td>
<td>1) unsupervised ML approach</td>
</tr>
<tr>
<td></td>
<td>connectome project</td>
<td>Network</td>
<td></td>
<td>2) no specialized “autoencoder”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>layer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3) a composite network that</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>comprises two subnetworks</td>
</tr>
<tr>
<td>2</td>
<td>Genomics/predicting the function of</td>
<td>Recurrent Neural</td>
<td>Supervised</td>
<td></td>
</tr>
<tr>
<td></td>
<td>non-coding DNA (~98%)</td>
<td>Network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Genomics/clustering of cancer samples</td>
<td>Auto-encoder</td>
<td>Unsupervised</td>
<td></td>
</tr>
<tr>
<td></td>
<td>based on gene expression (~2%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Bioimage synthesis/ developmental</td>
<td>Generative Adversarial</td>
<td>Unsupervised</td>
<td></td>
</tr>
<tr>
<td></td>
<td>biology</td>
<td>Network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Small drug molecule design</td>
<td>Reinforcement Learning</td>
<td>Reinforcement</td>
<td></td>
</tr>
</tbody>
</table>
Basic autoencoders: a simple example

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

Transformation performed by a layer: \( T_{out} = A(W \cdot T_{in} + b) \)

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)
- layer activations:
  - linear (Model 1) and sigmoid (Model 2)
- dimension of the latent space (=2)
- sizes of tensors (X, Y: 9; X₁, Z₁: 16)
The training code for basic Model 1

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
The prediction code for basic Model 2

encoder, decoder, load_weights, predict, pyplot

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


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

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

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

**Programs:**
- `download_data.sh`
- `process_data.py`
- `tybalt_train.py`
- `tybalt_predict.py`
- `tybalt_visualize.R`

**Files:**
- Raw data (TSV)
- Pre-processed data (TSV)
- Checkpoints (HDF5)
- Encoded data; tSNE features (TSV)
- Plots (PNG)
An overview of the Tybalt code

**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
- VAE loss function

**Running the model**
- fit
- predict
- perform_tSNE
- more on GD-based optimization

```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, orig_dim, latent_dim, \
    hidden_dim, beta, validation_data = get_data()

    # Define model
    encoder, decoder, combined_model = build_combined_model(opt.model_name, orig_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])
    else:
        combined_model.compile(optimizer=optimizers.Adadelta(lr=opt.learning_rate), loss='mse')

    # Run the model
    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=validation_data, \
                           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, callbacks=[checkpointer], \
                           validation_data=(rnaseq_test_df, rnaseq_test_df))
    encoded_df = encoder.predict_on_batch(rnaseq_df)
    if opt.model_name == 'vae':
        encoded_df = encoded_df[2]
    encoded_df = pd.DataFrame(encoded_df, index=rnaseq_df.index)
    perform_tSNE(encoded_df, opt.model_name)
```
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
(downloaded)

Number of genes = 20,530

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

Number of genes = 5,000

HiSeqV2 (TSV)

process_data.py

pancan_scaled_rnaseq.tsv

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
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)

Defaults:
- original_dim = 5,000
- latent_dim = 100

Reconstruction loss = MSE({X_i}, {X'_i})
The VAE (variational autoencoder) model

Reconstruction loss

Reparametrization trick:
\[ z = \mu_i + \sigma_i \cdot \epsilon \]
\[ \epsilon = N(0,1) \]

Regularization loss

High-dim reconstruction of input data

Defaults:
original_dim = 5000
hidden_dim = 100
latent_dim = 100

https://arxiv.org/abs/1312.6114
Why “Variational”? Computing the VAE loss.

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

Deterministic approach

Objective: \( \text{loss}(X, X') \rightarrow \min \)

\[
\log P(X) = \log \int P(X|z) P(z) \, dz \rightarrow \max
\]

“evidence”

apply the Bayes rule; replace integration with sampling

\[
\log P(X) = \mathbb{E}_{z \sim P(z|X)} P(X|z) - D_{KL}[P(X|z) \| P(z)] \rightarrow \max
\]

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

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

Probabilistic approach

Objective: \( P(X) \rightarrow \max \)

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

Keras implementation

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

“evidence lower bound” (ELBO)

“evidence” always \(\geq 0\)

Reconstruction term

Regularization term

Reconstruction loss

Regularization loss

BC = Binary cross-entropy
**tSNE: t-Distributed Stochastic Neighbor Embedding**

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


**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 all the neighborhoods from the high-dim to the low-dim space

For high-dim space:
Gaussian distribution

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

For low-dim space:
Student t-distribution (tSNE)

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

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

```
sinteractive sinteractive --mem=16g \ 
   --gres=gpu:v100,lscratch:10
module load tybalt
download_data.sh
process_data.py
tybalt_train.py -m vae
tybalt_predict.py -m vae
tybalt_visualize.R -v

17,44 All
```

VAE model + tSNE

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

- basic GD formula; $\gamma$ = learning rate
re-written: $\Delta w_t = w_{t+1} - w_t$

- small $\gamma$ → slow convergence along the valley
- larger $\gamma$ → oscillations in the perpendicular dir.

- gradient descent formula with momentum $\mu$ (usually, $= 0.9$)
- gradient descent formula with momentum $\mu$ and Nesterov accelerated gradient

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

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

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

http://ruder.io/optimizing-gradient-descent
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
How to run the basic AE code on Biowulf?

```
[user@biowulf]$ sinteractive --gres=gpu:v100:1
[user@cn4464]$ module load DLByExample/class3
... [+] Loading Tybalt 20190607 ...
[+] Loading DLByExample class3 ...
[user@cn4464]$ basic_ae_1h_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
...

[user@cn4464]$ basic_ae_1h_predict.py
...
[user@cn4464]$ basic_ae_3h_train.py
...
[user@cn4464]$ basic_ae_3h_predict.py
...
```