Deep Learning by Example on Biowulf

Class #2: Analysis of biological sequence data using RNNs and 1D-CNNs

Gennady Denisov, PhD
Class #2 Goals

**DL networks to be discussed:**
- Recurrent Neural Networks (RNNs)
- 1D Convolutional Neural Networks (1D-CNNs)

**Standard non-bio RNN benchmark:**
**IMDB movie review sentiment prediction:**

**Popular non-bio applications:**
- natural language processing
- text document classification
- time series classification, comparison and forecasting
- ...

**Bio example #2:**
predicting a function of the non-coding DNA

```
ATTCCGTAATCTACGATTAAGTCACAACCAAAACC
```

```
[010011010100111010\ldots110]
```

**Motifs:**

```
CTCF  FOS  HNF4A  \\
CIS-BP CIS-BP CIS-BP

POU5F1  SNAI1  P63  \\
CIS-BP CIS-BP CIS-BP
```

**Motif database**

**Distinctive features of the biological example:**
1) a **vector** of binary labels is assigned to each data sample
2) need to identify the sequence **motifs**
Examples overview

Examples #1 and #2 are complementary one to another in a number if ways, including:

- different approaches to **building a network**: branched network (#1) => Functional API unbranched network (#2) => Sequential construct

- different approaches to **training a network**: limited ground truth data, labeled manually (#1) => augmentation plenty of the ground truth data, generated using NGS (#2) => no augmentation
**Input**: a set of training sequences of 0’s and 1’s and **binary** labels assigned to each sequence, depending on whether or not a certain motif is present in the sequence.

**Task**: predict the occurrence of the **unknown** motif in new sequences.

**Example**: 01011100101

\[ Y = \sum w_i X_i + b \]

Conv2D
- parameters: \( w_i, b \)
- hyperparameters:
  - \( f = \text{filter/kernel size} = (3) \)
  - padding (= “valid”)

Conv1D
- parallelizable
- memoryless
- independent channels
- output: all the channel units
- output shape: \((n, L-f+1)\)
- # params = \((f+1)n\)

SimpleRNN
- sequential
- has memory
- interacting channels
- output: only last units \(Y_{1,L}, \ldots, Y_{n,L}\)
- output shape: \((n, 1)\)
- # params = \(n + n + n^2n = 2n + n^2\)
The SimpleRNN training code
Sequential construct, SimpleRNN layer, motif, metrics

https://github.com/keras-team/keras

---

Header:
- general Python imports
- Dense, SimpleRNN
- Sequential

Get data
- generate “synthetic” data
  - a motif to search for
- training samples x_train and binary labels y_train

Define a model
- Sequential construct approach
  - compile, loss, optimizer, metrics

Run the model
- fit, checkpoint, epoch, callbacks
Two approaches to building models in Keras: the Functional API vs Sequential construct

Any / branched network (e.g. “mini-UNet”)

from keras.models import Input, Model
from keras.layers import L1, L2

# Define a model
X = Input(…)
Y = L1(X)
Z = L2[ X, Y ]
model = Model( inputs = X, outputs = Z )
model.compile(…)

The Functional API approach
- explicitly uses tensor names
- applicable to any type of networks, both branched or unbranched

Only unbranched / sequential newtwork

from keras.models import Sequential
from keras.layers import L1, L2

# Define a model
model = Sequential()
model.add(L1)
model.add(L2)
model.compile(…)

The Sequential construct approach
- does not explicitly use tensor names
- applicable only to unbranched networks
- a slightly shorter code
The SimpleRNN prediction code

**Header:**
- general python imports
- Dense, SimpleRNN
- Sequential

**Get data**
- generate “synthetic” data
- a motif to search for
- testing data

**Define a model**
- Sequential construct
- approach
- compile, loss, optimizer, metrics, summary

**Run the model**
- load_weights
- predict
How to run the SimpleRNN and Conv1D code on Biowulf?

```
$ sinteractive --gres=gpu:v100:1 --mem=4g

module load DLBio/class2

ls SDLBI0_BIN
conv1d_predict.py conv1d_train.py simplernn_predict.py simplernn_train.py

simplernn_train.py
Using TensorFlow backend.

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple_rnn_1</td>
<td>(None, 3)</td>
<td>15</td>
</tr>
<tr>
<td>dense_1</td>
<td>(None, 1)</td>
<td>4</td>
</tr>
</tbody>
</table>

Total params: 19
Trainable params: 19
Non-trainable params: 0

Epoch 1/1000
1000/1000 [====================] - 1s 561us/step - loss: 0.8053 - acc: 0.4820

Epoch 1000/1000
1000/1000 [====================] - 0s 62us/step - loss: 0.0061 - acc: 1.0000

simplernn_predict.py
Using TensorFlow backend.

y, y_test= 0 0
y, y_test= 1 1
y, y_test= 0 0
y, y_test= 1 1

conv1d_train.py
...

conv1d_predict.py
...
Example 2. **DanQ**: Predicting the function of noncoding DNA *de novo* from sequence


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

**Task:** predict the targets directly from DNA sequence

- **Events** (“targets”; total = 919 types):
  - transcription factor binding sites (690 types)
  - DNase I hypersensitivity sites (125 types)
  - histone marks (104 types)

**Models:**
- DeepSEA (2015) – Torch, **DanQ** (2016) – Keras,
- Basset (2016) – Torch, Basenji (2017) – Tensorflow,

**DL frameworks:**
- Torch, **Keras**, Tensorflow, …

**Network types:**
- CNN (1D), RNN, …
Overview of the DanQ training code
(only the main function is shown)

Imports statements, other function definitions

Header
- parse the command line options

Get data
- training, testing and validation data

Define a model
- DanQ model
- DeepSEA model
- LSTM layer
- Dropout layer
- multi_gpu_model
- compile, loss, metrics
- optimizer

Run the model
- Easlystopping
- fit
Available data
one-hot encoding; training, validation, and testing data

**One-hot encoding:**

ATTCCGTAATCTAGATTAAGTCACAACCAAAACC

Training data: \( N = 4.4 \text{ M} \) => adjust **parameters**
Validation data: \( N = 8 \text{ K} \) => tune **hyperparameters**
Testing data: \( N = 455 \text{ K} \) => test **predicted labels**

Sequence length (=1000 bp)

Num. targets (=919)

X (data): \( 4 \times 1000 \times N \)

y (labels): \( 919 \times N \)
An overview of the DanQ model
LSTM, BLSTM, MaxPooling1D, Dropout


Input

Output

One hot encoding

Conv1D(320, 26, …)
# allows computing PWMs

Dropout(0.2)
MaxPooling1D(pool_size=13, …)

Bidirectional(LSTM(320, …))
# recurrent layer

Dropout (0.5), Flatten(), Dense(…)

Activation(“sigmoid”)  
Activation(“relu”), Dense(…)

BLSTM: # params = 1,640,960
Conv1D: # params = 33,600
**Long Short-Term Memory (LSTM) layer**

*S Hochreiter, J Schmidhuber, Neural computation 9, p.1735 (1997)  
https://adventuresinmachinelearning.com/keras-lstm-tutorial*

Why not SimpleRNN?  
- the “vanishing gradients” issue:  
  \[ \nabla J(w) \to 0 \]

---

**SimpleRNN**: one step  
1) \( X_t, Y_{t-1} \Rightarrow Y_t \)

**LSTM**: two steps  
1) \( X_t, Y_{t-1}, S_{t-1} \Rightarrow S_t \)  
2) \( X_t, Y_{t-1}, S_t \Rightarrow Y_t \)

\[
G(X_t, Y_{t-1}) = \tanh(b_G + w_{XG} \cdot X_t + w_{YG} \cdot Y_{t-1}) \\
I(X_t, Y_{t-1}) = \sigma(b_I + w_{XI} \cdot X_t + w_{YI} \cdot Y_{t-1}) \\
F(X_t, Y_{t-1}) = \sigma(b_F + w_{XF} \cdot X_t + w_{YF} \cdot Y_{t-1}) \\
E(X_t, Y_{t-1}) = \sigma(b_E + w_{XE} \cdot X_t + w_{YE} \cdot Y_{t-1})
\]

\( \times \) = elementwise multiplication; \( \sigma \) = sigmoid activation

# parameters = \( 4(2n + n^2) \)
Gradient descent-based optimizers: Batch GD, SGD and Mini-batch GD

\[ w_{t+1} = w_t - \gamma \cdot \nabla_w J(w_t; x, y) \]

- gradient descent formula for updating weights

\( w \) = vector of weights
\( t \) = update #
\( \gamma \) = learning rate
\( \nabla_w J \) = gradient of the loss with respect to weights
\( (x, y) \) = one data sample (= data item \( x \) + label \( y \))

Vanilla, a.k.a. “Batch” Gradient Descent:
- inefficient / not used for large training datasets
- average \( \nabla_w J(w_t; x, y) \) over all \( N \) samples in the training dataset
- perform one update of weights per epoch

Stochastic Gradient Descent:
- objective fluctuates, this may complicate convergence
- compute \( \nabla_w J(w_t; x^{(k)}, y^{(k)}) \) on a random sample \((k = 1, 2, \ldots, N)\)
- perform \( N \) updates per epoch by default, or as specified by the keyword argument steps_per_epoch in the method fit

Mini-batch Gradient Descent:
- \( \nabla_w J(w_t; x, y) \) averaged over a mini-batch
- \( N / \text{batch}\_\text{size} \) updates per epoch by default (where the \text{batch}\_\text{size} is a hyperparameter), or as specified by the keyword argument steps_per_epoch in the method fit
How to run the DanQ code on Biowulf?

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

Using a single GPU:

```
sinteractive --mem=64g --gres=gpu:v100:1,1scratch:100 \
    --cpus-per-task=14
module load danq
ls $DANQ_SRC
danq_predict.py danq_visualize.py models.py
```

Using 4 GPUs:

```
sinteractive --mem=64g --gres=gpu:v100:4,1scratch:100 \
    --cpus-per-task=14
module load danq
ls $DANQ_SRC
danq_predict.py danq_visualize.py models.py
```

Visualizations:

- ROC curve:

```
   True positive rate (TPR) 0.0 0.2 0.4 0.6 0.8 1.0
   False positive rate (FPR) 0.0 0.2 0.4 0.6 0.8 1.0
   ROC curve
```

- Motif sequence logo:

```
    TGAATTC
    TACCCTG
```

Models: DanQ, DeepSEA

# DeepSEA model:

Conv1D(320, 8, ...)
MaxPooling
Conv1D(480, 8, ...)
MaxPooling
Conv1D(960, 8, ...)
Flatten
Dense(919, ...)
Activation("sigmoid")
Conclusions

1) Further intro using simple examples
   - SimpleRNN vs Conv1D layers/transformations
   - the notion of the RNN network memory and interacting channels
   - Functional API vs Sequential approach to building Keras models
   - metric, model summary and the # of parameters used by layers

2) Predicting the function of a non-coding DNA
   - the DanQ and DeepSEA models
   - (Bidirectional) LSTM, MaxPooling 1D and Dropout layers
   - how to run the DanQ code on Biowulf

3) Gradient descent-based optimizers:
   - SGD and Mini-batch DG
BACKUP SLIDES
Modelling transcription factor binding sites and spacing with 1D CNNs

G. Eraslan et al, Nature Reviews Genetics 2019
Predicting motifs

David R. Kelley et al - Basset: …, Genome Res, 2016, 26:990–999

Overall, 45% of filters could be annotated