An Introduction to Deep Learning with RapidMiner

Philipp Schlunder - RapidMiner Research
What’s in store for today?

1. Things to know before getting started

2. What’s Deep Learning anyway?

3. How to use it inside RapidMiner
Before getting up to speed
What do I actually need?

Hardware

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Recommended Starter Kit</th>
<th>Optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC or Mac running RapidMiner</td>
<td>PC with supported NVIDIA GPU running Linux/Windows</td>
<td>Dedicated Server with multiple GPUs running Linux</td>
</tr>
</tbody>
</table>

Software

- **RapidMiner Studio** or **RapidMiner Server**
- **Python** installation (at least Version 3.5) with
  - **Pandas**
  - Linux recommended: **Tensorflow** (for GPUs)
  - Windows recommended: **Microsoft CNTK**
- **Keras** extension
I don’t have a server with GPUs!

We’ve got you covered with **RapidMiner Server** on the **Amazon AWS** and **Microsoft Azure** marketplaces

- Put your server where your data is
- Server installation in minutes
- No need for own hardware
- Bring-your-own-license images (Amazon AMIs & Microsoft VHDs)
When should I try Deep Learning?

1. Stuck with current model
   - Accuracy not high enough
2. Huge amount of training data available
3. Fixed specific use-case
4. Need notion of memory
   - Understanding context
5. Multi-dimensional input
   - Pictures: 3 colors per Pixel
   - Text: Word Vectors
Things to consider

• Large number of parameters to tune
  → often req. huge amount of data
  – Is my infrastructure suited for that?
  – Is the potential ROI high enough for the resource investment?
• For many general ML tasks only minor improvement
• Often not that flexible and very domain specific
• Hard to understand reasoning (consider LIME)
• General Data Protection Regulation (25\textsuperscript{th} of May 2018, EU-based)
  – “right to an explanation”
  – “prevent discriminatory effects based on race, opinions, health”
What is Deep Learning anyway?
Machine Learning
Hierarchical Learning
Hierarchical Learning

Input

First Representation

Second Representation

Retrieving Data

Split Data

Apply Model

Set Role

Split Data (2)

Apply Model (2)
Hierarchical Learning
Deep Learning
Layers

Input Layer

Hidden Layer

Output Layer
Neurons

Input Layer

Number of Attributes

Hidden Layer

Output Layer
Neurons

Input Layer
Number of Attributes

Hidden Layer

Output Layer
Number of Label values
**Neurons**

- **Input Layer**: Number of Attributes
- **Hidden Layer**: New Attribute based on previous ones and some activation function (e.g., ReLu)
- **Output Layer**: Number of Label values

New Attribute based on previous ones and some activation function (e.g., ReLu)
Layer: Fully-Connected

Input Layer

Hidden Layer

Fully-Connected (Dense) Layer

Output Layer
Change of Perspective
Layer: Convolutional
Layer: Convolutional
Layer: Convolutional

\[ \text{Activation Map} = \frac{(N - F)}{\text{Stride}} + 1 \]

- \( N = 4 \)
- \( F = 2 \)

Example Filter
Layer: Convolutional

\[ X = \frac{(N - F)}{\text{Stride}} + 1 \]

- \( N = 4 \)
- \( F = 2 \)

Activation Map

\((N - F) / \text{Stride} + 1\)
Layer: Convolutional

N = 4

F = 2

Stride = 1

Activation Map

$$\frac{N - F}{\text{Stride}} + 1 = \frac{4 - 2}{1} + 1 = 3$$
Layer: Convolutional

Example Filter

Activation Map

4x1 + 3x0 + 1x0 + 7x1 = 11
Layer: Convolutional

Example Filter

Activation Map
Layer: Convolutional

Example Filter

Activation Map

\[ X \times \text{Example Filter} = \text{Activation Map} \]
Layer: Convolutional

Example Filter

X

Activation Map
Layer: Convolutional

Example Filter

Activation Map

4  3  8  7
1  7  8  1
9  6  3  2
3  4  0  4

X

1  0
0  1

11  11  9
7  10  10
13  6  7
Layer: Convolutional

Padding = 1
Layer: Convolutional

Activation Map
Layer: Convolutional

Filter:

1 0
0 1

Activation Map
Layer: Convolutional

Filter:

1  0
0  1

1  1
0  0

Convolutional Layer

Activation Map
Layer: Pooling

Max(imum) Pooling
Layer: Pooling

Stride = 2

Max(imum) Pooling
Layer: Pooling

Max(imum) Pooling
Layer: Pooling

Max(imum) Pooling
Layer: Pooling

Max(imum) Pooling

Average Pooling

(4 + 3 + 1 + 7) / 4
Layer: Pooling

Max(imum) Pooling

- Representation becomes smaller → less values to optimize
- Applied to each activation map of the previous layer
Layer: Recurrent

State of the layer
Layer: Recurrent

State of the layer changes with each repetition
Layer: Recurrent

I drank some coffee

Drinking a hot beverage that [...] for me it is coffee.

Neurons with hidden states

Layer: Recurrent

I drank some coffee

Drinking a hot beverage that [...] for me it is coffee.

Neurons with hidden states
Layer: Long Short Term Memory

Information can be:
- Not stored (input)
- Only stored partially (gate)
- Forgotten
- Be considered only partially (output)
Layer: Dropout

Dropout is a form of regularization. It helps reduce complexity and frame the model.

**Dropout Rate**
- Probability of dropping a neuron
- Between 0 and 1
Input & Setup
Input Layer

Hidden Layer

Output Layer
input_shape depends on layers used

- Convolution/Recurrent:
  \[(\text{timesteps}, \text{input\_dim})\]
  \[= (1, 3)\]

- Others:
  \[(\text{input\_dim},)\]
  \[= (3,)\]
Input

**input_shape**

- Convolution/Recurrent: 
  
  $$(\text{timesteps}, \text{input\_dim})$$ 
  
  $$= (4, 3)$$ 
  
  since the data contains timesteps
Input

Input Data Set

Entry (picture, sentence, ...)

Batch

batch_size
- Number of entries to use in one model building step (obtaining weights)
Here: 4
The Model – Weights

Weight
amount of a neuron’s contribution

Weights are changed using an optimizer

An optimizer reduces the loss

One weight change cycle is called an epoch
### The Model – Building it

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Binary Classification</th>
<th>Multi-Class Classification</th>
<th>Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimizer</td>
<td>adam/rmsprop</td>
<td>adam/rmsprop</td>
<td>adam/rmsprop</td>
</tr>
<tr>
<td>loss</td>
<td>binary_crossentropy</td>
<td>categorical_crossentropy</td>
<td>mse</td>
</tr>
</tbody>
</table>

While training observe loss (log/Tensorboard):

- Bump $\rightarrow$ change weight init
- Lowers slowly $\rightarrow$ learning rate too low
- Fast decline, then stagnating $\rightarrow$ learning rate too high
- Brief decline, rises extremely afterwards $\rightarrow$ learning rate way too high

- `rmsprop`: esp. good choice for recurrent
- `adam`: rmsprop with extra (momentum)
The Model – Architecture

- Start small and simple
- Test overfitting on small batch
  (small loss, ~1.0 accuracy)
  - Add regularization until loss goes down
  - Not going down, learning rate too low
  - Constant or NaN, learning rate too high

- Hyper Parameter Tuning
- Often:
  - No. filters rises (64 → 128 → 256)
  - Filter size shrinks (5 → 3)
  - Stride shrinks (2 → 1)
  - Padding is introduced (0 → 1)
Deep Learning Summary
Summary

• Different Types of Layers:
  – Fully-connected, convolutional, recurrent, pooling, dropout, ...

• Model is defined by:
  – Setup of layers (architecture)
  – Weights obtained through optimization of a loss over several epochs

• Classification vs. Regression:
  – Change number of units in last layer (Number of possible classes vs. number of targeted values to predict)
  – Change loss from crossentropy to mse
How to use it in RapidMiner?
## Options in RapidMiner

<table>
<thead>
<tr>
<th>Feature</th>
<th>Neural Net</th>
<th>Deep Learning H20</th>
<th>Deep Learning Keras</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully-Connected Layer</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Basic Optimization</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Advanced Optimization</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Multi-.Threading</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>GPU Support</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Advanced Layers</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Complexity</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Demo Time
Classification
Classification

- **4 attributes (a1 – a4)**
- **3 label values → multi-class classification**

**Parameters**

- **input shape**: (4,)
- **loss**: categorical_crossentropy
- **optimizer**: Adam
- **learning rate**: 0.001
- **beta 1**: 0.999
- **beta 2**: 0.999
- **epsilon**: 1.0E-8
- **decay**: 0.0
- **epochs**: 128
- **batch size**: 32

©2017 RapidMiner, Inc. All rights reserved.
Classification

Parameters

- **layer type**: Dense
- **no units**: 8
- **activation function**: 'relu'
- **use bias**: checked
- **kernel initializer**: glorot_uniform(seed=None)
Classification

3 label values
Regression

Parameters

Windowing

- series representation: encode_series_by_exam...
- window size: 30
- step size: 1

- create single attributes
- create label
- add incomplete windows
- stop on too small dataset
Regression

Parameters

- Windowing
  - series representation: encode_series_by_example
  - window size: 30
  - step size: 1
- create single attributes
- create label
- add incomplete windows
- stop on too small dataset

Keras Model

- input shape: (30, 1)
- loss: mean_squared_error
- optimizer: Adam
- learning rate: 0.001
- beta 1: 0.999
- beta 2: 0.999
- epsilon: 1.0E-8
- decay: 0.0
- epochs: 256
- batch size: 32

©2017 RapidMiner, Inc. All rights reserved.
Regression

32 entries

256 iterations of changing/optimizing the weights with a method called ‘Adam’
Regression
Regression

- **Parameters**
  - **Add Convolutional Layer**
    - layer type: Conv1D
    - filters: 64
    - kernel size 1d: 2
    - strides 1d: 1
    - padding: valid

- **Diagram**
  - 64 filters → 64 activation maps
  - Network architecture includes:
    - Add Convolutional Layer
    - Add Pooling Layer
    - Add Convolutional Layer
    - Add Pooling Layer (2)
    - Add Core Layer
    - Add Core Layer (2)
    - Add Core Layer (3)
    - Add Core Layer (4)
    - Dropout Layer
    - Flatten
    - Fully-Connected (Dense)
    - Output Layer

©2017 RapidMiner, Inc. All rights reserved.
Regression

Parameters

- Add Pooling Layer
  - layer type: MaxPooling1D
  - pool size 1d: 2
  - strides 1d: 2
  - padding: valid

Diagram:
- Add Convolutional Layer
- Add Pooling Layer
- Add Convolutional Layer
- Add Pooling Layer (2)
- Add Core Layer
- Add Core Layer (2)
- Add Core Layer (3)
- Add Core Layer (4)
- Dropout Layer
- Flatten
- Fully-Connected (Dense)
- Output Layer
Regression

Parameters

- **layer type**: Dropout
- **rate**: 0.25
- **noise shape**: None
- **seed**: None

Diagram showing a neural network with layers such as Add Core Layer, Add Convolutional Layer, Add Pooling Layer, Flatten, and Fully-Connected Layer.
Regression

Parameters

- Add Core Layer (2) (Add Core Layer)

layer type: Flatten

Add Convolutional Layer

Add Pooling Layer

Add Convolutional Layer

Add Pooling Layer (2)

Add Core Layer

Add Core Layer (2)

Add Core Layer (3)

Add Core Layer (4)

Dropout Layer

Flatten

Fully-Connected (Dense)

Output Layer
Regression

Parameters

- **layer type**: Dense
- **no units**: 250
- **activation function**: 'relu'
- **Use bias**: 
- **Kernel initializer**: glorot_uniform(seed=None)

250 neurons
Regression

1 value to estimate (closing date of 30th timestep)
Regression

Log storage directory
Regression

Start Tensorboard with log storage directory location provided through command line

tensorboard --logdir=C:/Users/PhilippSchlunder/Desktop/logs/

View loss graph in Browser under the address: Localhost:6006
Sources

- Stanford CS231n
- Keras docu
- When not to use deep learning
- Deep Learning is not the AI future