DKpro and Deep Learning with Java, part 1

PU Machine learning mit Java, Weka und UIMA

Nils Reiter,
nils.reiter@uni-koeln.de

January 26, 2021
Winter term 2020/21

Section 1

Looking Back

Exercise 10

https://github.com/idh-cologne-machine-learning-mit-java/exercise-10

Section 2

DKpro

DKpro

- Existing NLP tools re-packaged as a UIMA component
- Import/installation via maven dependencies
- `https://dkpro.github.io/dkpro-core`

DKpro

- Existing NLP tools re-packaged as a UIMA component
- Import/installation via maven dependencies
- `https://dkpro.github.io/dkpro-core`

Examples

- Berkeley Parser
- (Stanford) CoreNLP
- LangDetect language identifier
- Mallet LDA Topic Model Trainer/Inferencer
- Mate Tools Semantic Role Labeler
- ...

DKpro

Example

Listing 1: An entire NLP pipeline from plain text to semantic roles

```
1  SimplePipeline.runPipeline(
2    CollectionReaderFactory.createReaderDescription(TextReader.class,
3      TextReader.PARAM_SOURCE_LOCATION, "DIRECTORY"),
4    AnalysisEngineFactory.createReaderDescription(LanguageIdentifier.class),
5    AnalysisEngineFactory.createReaderDescription(BreakIteratorSegmenter.class),
6    AnalysisEngineFactory.createReaderDescription(MatePosTagger.class),
7    AnalysisEngineFactory.createReaderDescription(CoreNlpDependencyParser.class),
8    AnalysisEngineFactory.createReaderDescription(LingPipeNamedEntityRecognizer.class),
9    AnalysisEngineFactory.createReaderDescription(ClearNlpSemanticRoleLabeler.class),
10   AnalysisEngineFactory.createReaderDescription(JsonWriter.class,
11     JsonWriter.PARAM_TARGET_LOCATION, "output")
12 )
```

- Mixing components is fine, but pay attention to tag set compatibility

Section 3

Deep Learning

Introduction

Multiple, overlapping trends ('deep learning' is a buzz word)

Introduction

Multiple, overlapping trends ('deep learning' is a buzz word)

- Distributed representations: Input instances as vectors
  - No 'manual' feature extraction
  - Instances are 'embedded' into high-dimensional vector space
    - Distributed: Embedding incorporates information from contexts of multiple instances
  - More training data
  - Embeddings are learned from data, therefore: 'representation learning'

Introduction

Multiple, overlapping trends ('deep learning' is a buzz word)

- Distributed representations: Input instances as vectors
  - No 'manual' feature extraction
  - Instances are 'embedded' into high-dimensional vector space
    - Distributed: Embedding incorporates information from contexts of multiple instances
  - More training data
  - Embeddings are learned from data, therefore: 'representation learning'
- (Artificial) Neural networks: Classification algorithms (first described in the 1940s!)
  - Neuron: Building block
  - Takes input from previous neurons, applies mathematical function, outputs to next function

Neural networks

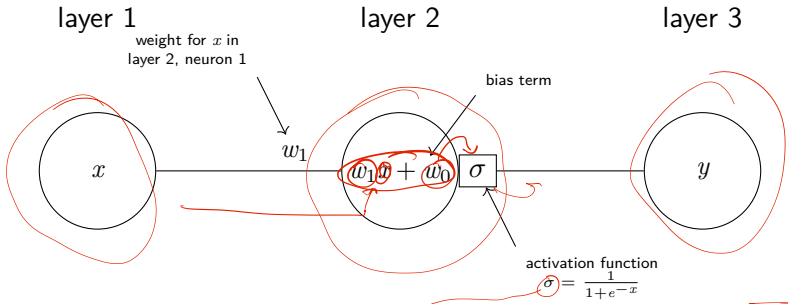
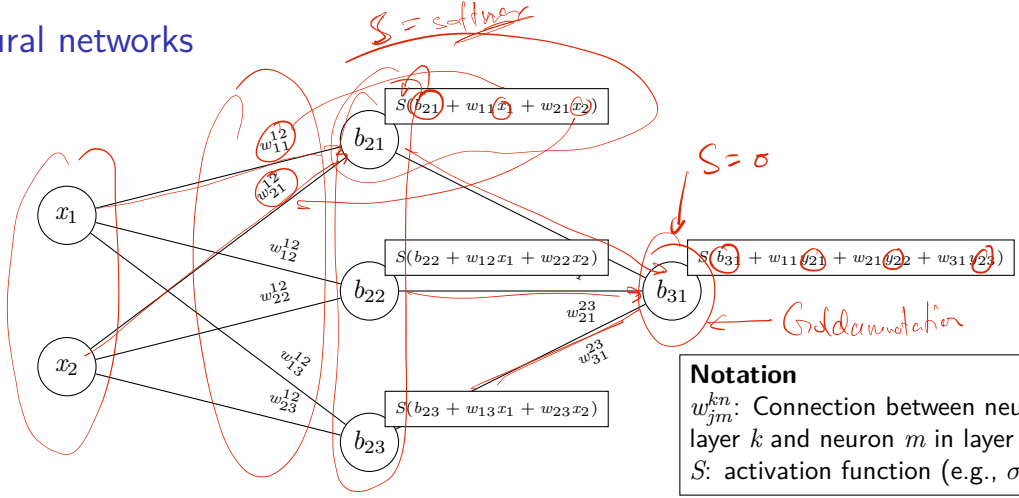


Figure: 1 neuron = logistic regression (with 1 feature)

Neural networks



Notation

w_{jm}^{kn}: Connection between neuron j in layer k and neuron m in layer n

S: activation function (e.g., σ)

Figure: A simple neural network with 1 hidden layer

Forward pass (= prediction model)

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)

Forward pass (= prediction model)

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)
- Practically, a lot of the computation happens in matrices
  - Hidden layer
    - Weights from input to hidden: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$

Linear Algebra

Forward pass (= prediction model)

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)
- Practically, a lot of the computation happens in matrices
  - Hidden layer
    - Weights from input to hidden: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$
- Hidden layer computation
  - $f_2(X) = \sigma((W_{1,2}^T X) + B_2)$

Forward pass (= prediction model)

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)
- Practically, a lot of the computation happens in matrices
  - Hidden layer
    - Weights from input to hidden: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$
- Hidden layer computation
  - $f_2(X) = \sigma((W_{1,2}^{\intercal} X) + B_2)$
- Deep learning involves a lot of matrix multiplication
  - GPUs are highly optimized for this
  - Hint: Gaming-GPUs that support CUDA are also usable for deep learning

Forward pass (= prediction model)

- If we have all the weights, bias terms, numbers of neurons and layers, we can compute the output of the network
  - Conceptually: Applying functions in sequence: $y = f_3(f_2(f_1(x)))$ (one per layer)
- Practically, a lot of the computation happens in matrices
  - Hidden layer
    - Weights from input to hidden: $W_{1,2} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
    - Biases $B_2 = (b_{21}, b_{22}, b_{23})$
- Hidden layer computation
  - $f_2(X) = \sigma((W_{1,2}^{\intercal} X) + B_2)$
- Deep learning involves a lot of matrix multiplication
  - GPUs are highly optimized for this
  - Hint: Gaming-GPUs that support CUDA are also usable for deep learning
- Numeric input and output

Feedforward Neural Networks

- The above is called a "feedforward neural network"
  - Information is fed only in forward direction
- Design choices
  - Activation function: All neurons of one layer have the same

sigmoid $y = \sigma(x) = \frac{1}{1+e^{-x}}$ – 'squashes' everything to a value between 0 and 1
relu $y = \max(0, x)$ – Makes everything negative to 0
softmax Scales a vector such that values sum to 1 (probability distribution)

  - Layer size: Number of neurons in each layer

Practical Deep Learning

- Hardware acceleration is important for efficiency
- Python
  - Important developments in neural networks
  - Mainstream language: Documentation, expertise, manuals, ...
  - Libraries: TensorFlow/keras, NumPy, scikit-learn, pandas (the big four)
    - Tensorflow has a C++ core

Practical Deep Learning

- Hardware acceleration is important for efficiency
- Python
  - Important developments in neural networks
  - Mainstream language: Documentation, expertise, manuals, ...
  - Libraries: TensorFlow/keras, NumPy, scikit-learn, pandas (the big four)
    - Tensorflow has a C++ core
  - Use Python for NLP ☺

Practical Deep Learning

- Hardware acceleration is important for efficiency
- Python
  - Important developments in neural networks
  - Mainstream language: Documentation, expertise, manuals, ...
  - Libraries: TensorFlow/keras, NumPy, scikit-learn, pandas (the big four)
    - Tensorflow has a C++ core
  - Use Python for NLP ☺

Extract, Transform, Load (ETL)

- Feature extraction → ETL
- Better data representation → better results
- Interesting experiments are not about, e.g., layer size, but about data representation
  ⇒ ETL pipeline is important

Libraries

- Data representation: ND4J
- ETL library: Datavec
  - Stores n-dimensional arrays *outside* of the Java heap
  - "API mimics the semantics of Numpy, Matlab and scikit-learn"
  - libnd4j: C++ part
- Neural networks: Deeplearning4J (DL4J)
  - Can run on CUDA (GPU), partially written in C++
  - Complexity sources
    - Data sets that don't fit in memory (unlike Weka)
    - Efficiency for productive use

Libraries

- Data representation: ND4J
- ETL library: Datavec
  - Stores n-dimensional arrays *outside* of the Java heap
  - "API mimics the semantics of Numpy, Matlab and scikit-learn"
  - libnd4j: C++ part
- Neural networks: Deeplearning4J (DL4J)
  - Can run on CUDA (GPU), partially written in C++
  - Complexity sources
    - Data sets that don't fit in memory (unlike Weka)
    - Efficiency for productive use
- Similar to Python

Libraries

- Data representation: ND4J
- ETL library: Datavec
  - Stores n-dimensional arrays *outside* of the Java heap
  - "API mimics the semantics of Numpy, Matlab and scikit-learn"
  - libnd4j: C++ part
- Neural networks: Deeplearning4J (DL4J)
  - Can run on CUDA (GPU), partially written in C++
  - Complexity sources
    - Data sets that don't fit in memory (unlike Weka)
    - Efficiency for productive use
- Similar to Python
- Plan
  - Today / Exercise 11: Data preparation
  - Next week / Exercise 12: Training an actual network

ND4J: n-Dimensional Arrays

int[][] = new int[] { new int[] {} }

- Terminology
  - Rank: Number of dimensions (n)
  - Shape: Size of each dimension
  - Length: Total number of elements (over all dimensions)
  - Views: Multiple NDArrays referring to the same underlying set of data

ND4J: n-Dimensional Arrays

- Terminology
  - Rank: Number of dimensions (n)
  - Shape: Size of each dimension
  - Length: Total number of elements (over all dimensions)
  - Views: Multiple NDArrays referring to the same underlying set of data

```
1 // create a matrix with 3 rows and 2 columns,
2 // filled with zeros
3 NDArray arr = Nd4j.zeros(3,2);
4
5 // create a 3D array
6 NDArray arr = Nd4j.ones(5,4,3);
```

ND4J: n-Dimensional Arrays

Operations

- Operations (addition, activation, ...) are executed via an `Executioner`-interface

```
1 NDArray arr = Nd4j.zeros(3,2); // [ [0, 0], [0, 0], [0, 0] ]
2 Nd4j.getExecutioner().exec(new ScalarAdd(arr, 5));
3 // arr = [ [5, 5], [5, 5], [5, 5] ]
```

ND4J: n-Dimensional Arrays

Operations

- Operations (addition, activation, ...) are executed via an `Executioner`-interface

```
1 NDArray arr = Nd4j.zeros(3,2); // [ [0, 0], [0, 0], [0, 0] ]
2 Nd4j.getExecutioner().exec(new ScalarAdd(arr, 5));
3 // arr = [ [5, 5], [5, 5], [5, 5] ]
4 EuclideanDistance dist = new EuclideanDistance(arr, Nd4j.ones(3,2));
5 Nd4j.getExecutioner().exec(dist);
6 Number n = dist.getFinalResult();
7 // n = 9.797959327697754
```

Datavec

- Fundamentals
  - Record: A single instance/entry
  - SequenceRecord: A single instance in a sequence of instances
  - RecordReader: Interface (with implementing classes) that reads records from some source
    - Sources: Image files, databases, CSV files, ...
    - Similar to iterators in plain Java
  - InputSplit: A list of loadable locations (for distributed corpora)

Datavec

- Fundamentals
  - Record: A single instance/entry
  - SequenceRecord: A single instance in a sequence of instances
  - RecordReader: Interface (with implementing classes) that reads records from some source
    - Sources: Image files, databases, CSV files, ...
    - Similar to iterators in plain Java
  - InputSplit: A list of loadable locations (for distributed corpora)

```
1 FileSplit fileSplit = new FileSplit(new File("src/main/resources/corpus"));
2 RecordReader reader = new FileRecordReader();
3 reader.initialize(fileSplit);
4
5 // this is usually done by the DL4J framework
6 while (reader.hasNext()) {
7   List<Writable> ds = reader.next();
8 }
```

Datavec

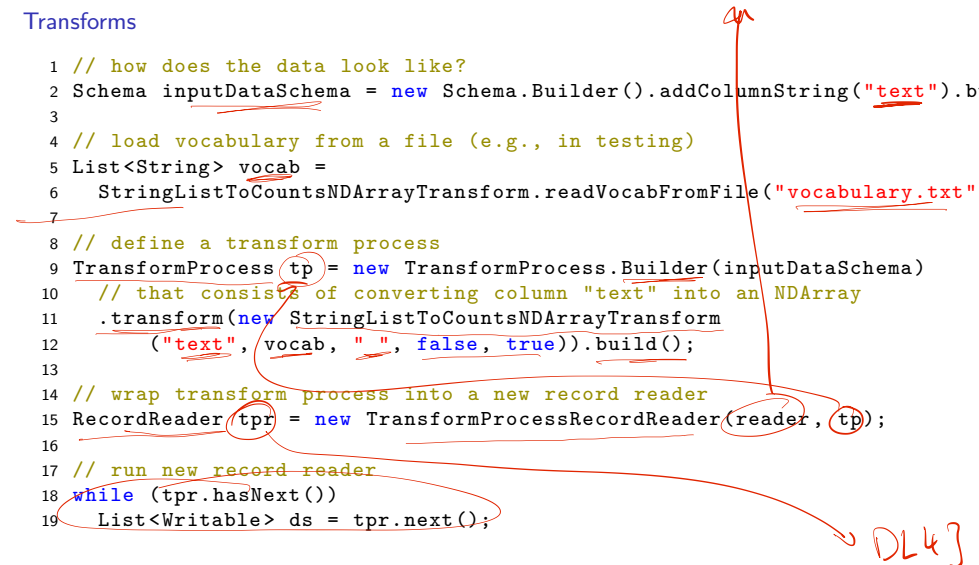
Transforms

- Transforms can be used to manipulate the data
- Use cases
  - Vectorising (e.g., text data into embeddings)
  - Normalisation and centralising (numerical data)
  - Removing, adding columns etc.
- This corresponds to filters in Weka

Datavec

Transforms

```
1  // how does the data look like?
2  Schema inputDataSchema = new Schema.Builder().addColumnString("text").build();
3
4  // load vocabulary from a file (e.g., in testing)
5  List<String> vocab =
6    StringListToCountsNDArrayTransform.readVocabFromFile("vocabulary.txt")
7
8  // define a transform process
9  TransformProcess tp = new TransformProcess.Builder(inputDataSchema)
10   // that consists of converting column "text" into an NDArray
11   .transform(new StringListToCountsNDArrayTransform
12       ("text", vocab, " ", false, true)).build();
13
14 // wrap transform process into a new record reader
15 RecordReader tpr = new TransformProcessRecordReader(reader, tp);
16
17 // run new record reader
18 while (tpr.hasNext())
19   List<Writable> ds = tpr.next();
```



demo

Section 4

Next Exercise

Exercise 11

https://github.com/idh-cologne-machine-learning-mit-java/exercise-11