Table of Contents

Introduction to DeepLearning4J

Introduction to Machine Learning

This section provides an introduction to Machine Learning and DeepLearning concepts.

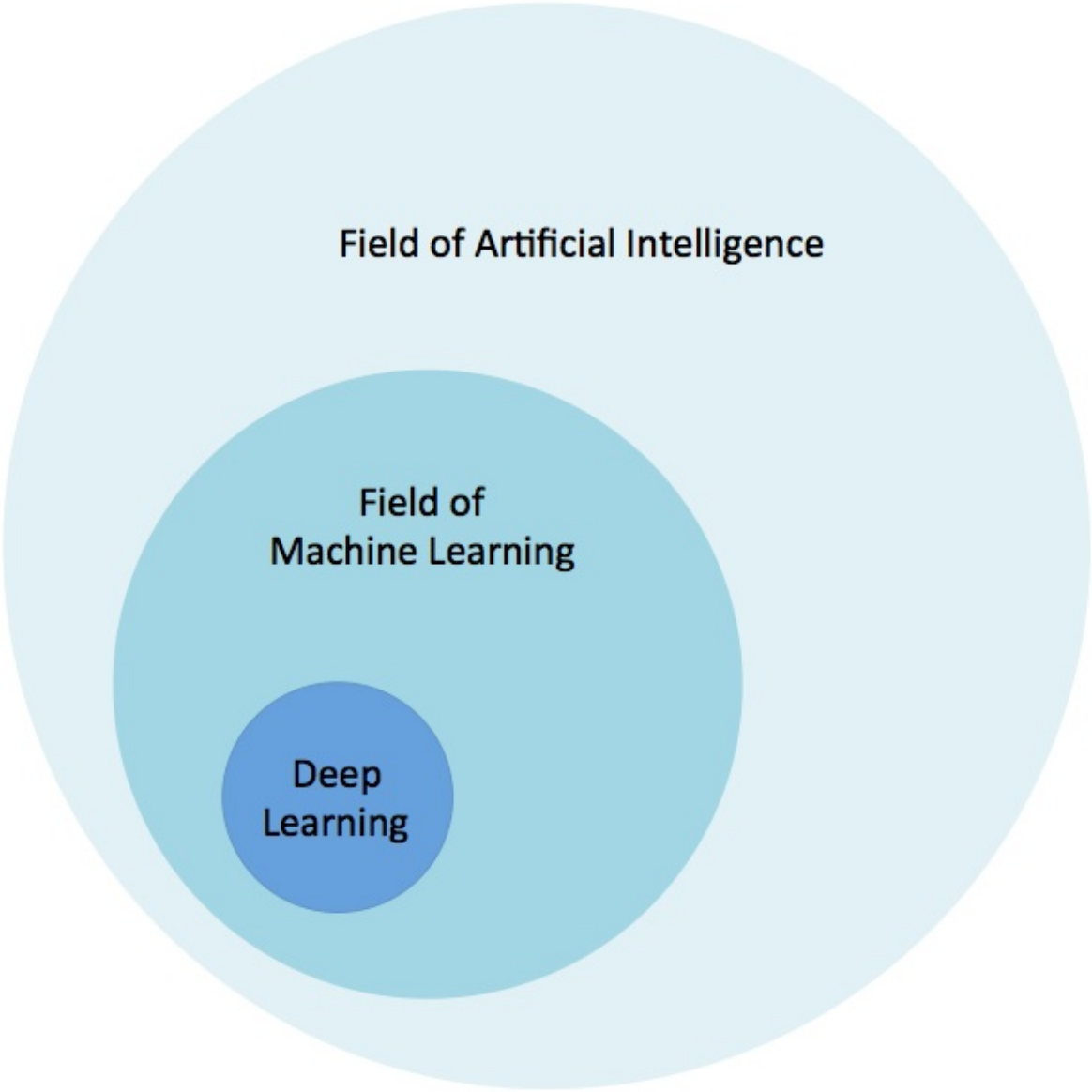
Introduction to Neural Networks with DeepLearning4J

Table of Contents

What is Machine Learning?

- Extracting Knowledge from raw data in the form of a model
    - Decision trees
    - Linear Models
    - Neural Networks
- Arthur Samuel quote
    - "Field of study that gives computers the ability to learn without being explicitly programmed"

A Diagram


Field of Artificial Intelligence
Field of
Machine Learning
Deep
Learning

Machine Learning Compared to Data Science/Mining

- Data Mining
  - The process of extracting information from the data
  - Uses Machine Learning
- Data Science
  - Data Mining from the lens of a statistician
  - Venn Diagrams
  - A way to get a raise
  - A more agreeable Actuary
  - A statistician using a Mac

Framing the Questions

- To build models we have to define
  - What is our training data (“evidence”)?
  - What kind of model (“hypothesis”) is appropriate for this data?
  - What kind of answer (“inference”) would we like to get from the model?
- These questions frame all machine learning workflows

Ax = b

- In neural networks we're solving systems of (non-linear) equations of the form
  - Ax = b
- A matrix
  - This is our set of input data converted into an array of vectors
- x vector
  - The parameter vector of weights representing our model
- b vector
  - Vector of output values or labels matching the rows in the A matrix

Ax = b Visually

Let’s Talk Linear Algebra

- Scalars
  - Elements in a vector
  - In compsci synonymous with the term “variable”
- Vectors
  - For a positive integer n, a vector is an n-tuple, ordered (multi)set, or array of n numbers, called elements or scalars
- Matricies
  - Group of vectors that have the same dimension (number of columns)

Solving Systems of Equations

- Two general Methods
  - Direct method
  - Iterative methods
- Direct method
  - Fixed set of computation gives answer
  - Data fits in memory
  - Ex: Gaussian Elimination, Normal Equations
- Iterative methods
  - Converges after a series of steps
  - Stochastic Gradient Descent (SGD)

Vectorization

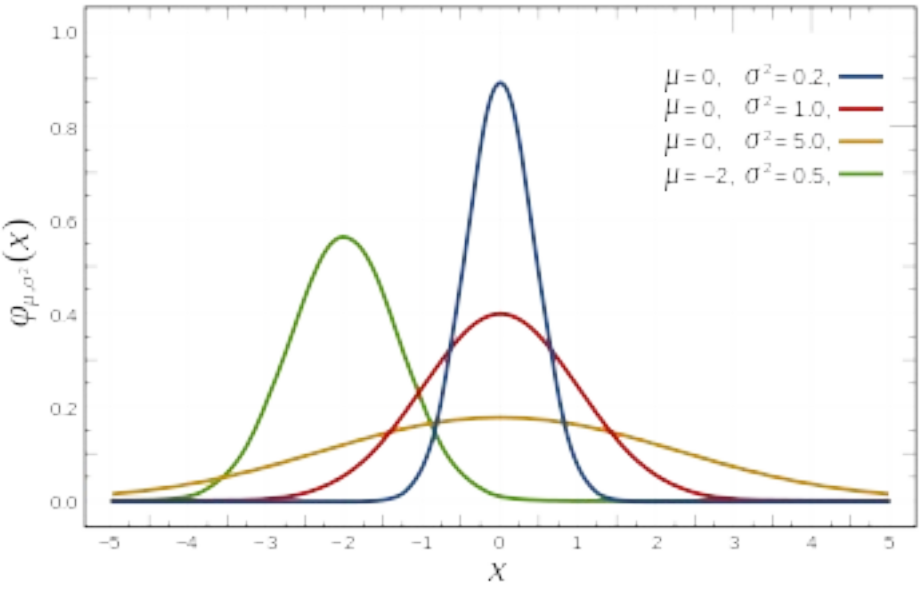
- To solve$Ax = b$ Ax = b with optimization methods such as SGD
  - We have to get raw data into the vectors and matrices
- This ends up being a lot of work
  - most folks never consider this phase to be *sniff* “real machine learning”
- Actually it's pretty key to building good models

Quick Statistics Review: Probability

- Probability
  - We define probability of an event E as a number always between 0 and 1.
  - In this context the value 0 infers that the event E has no chance of occurring and the value 1 means that the event E is certain to occur.
- The Canonical Coin Example
  - Fair coin flipped, looking for heads/tails (0.5 for each side)
  - Probability of sample space is always 1.0
  - P(Heads) = 0.5 every time

Probability Distributions

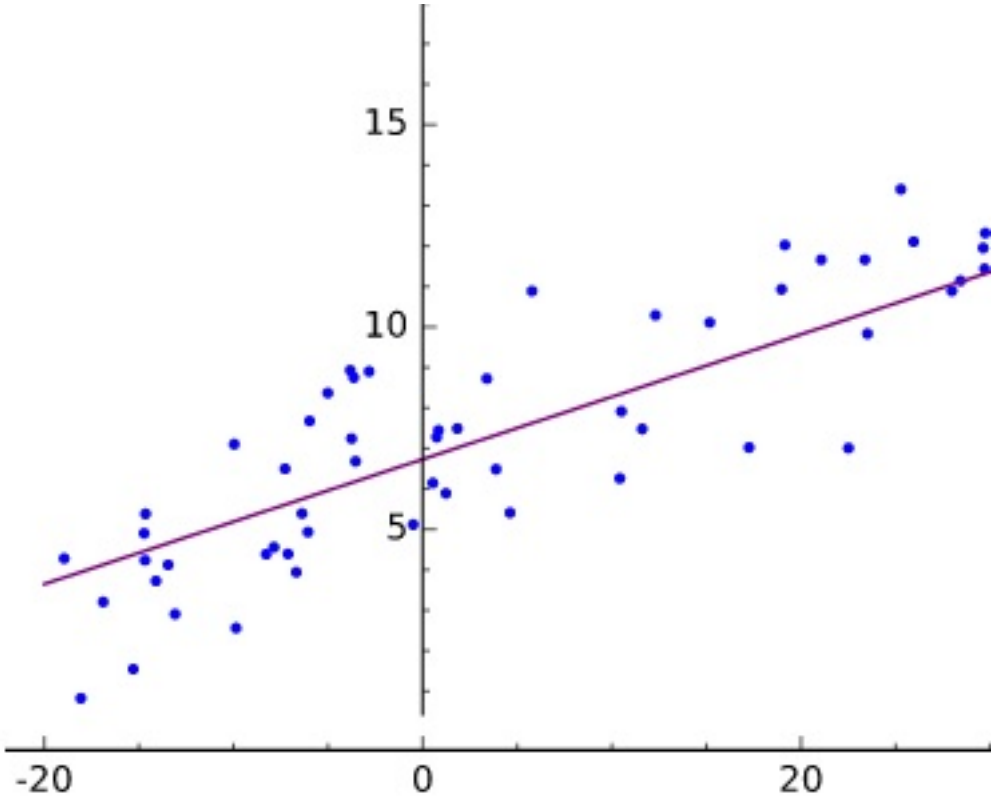
- A specification of the stochastic structure of random variables
- In statistics we rely on making assumptions about how the data is distributed
  - To make inferences about the data
- We want a formula specifying how frequent values of observations in the distribution are
  - And how values can be taken by the points in the distribution



High-Level Machine Learning

- Determine
  - Output desired (“question to be answered”)
  - Input data to build model (“evidence”)
  - Appropriate model (“hypothesis”)
- Setup data in Ax = b form
  - For linear models and neural networks
- Then Optimize the x parameter vector

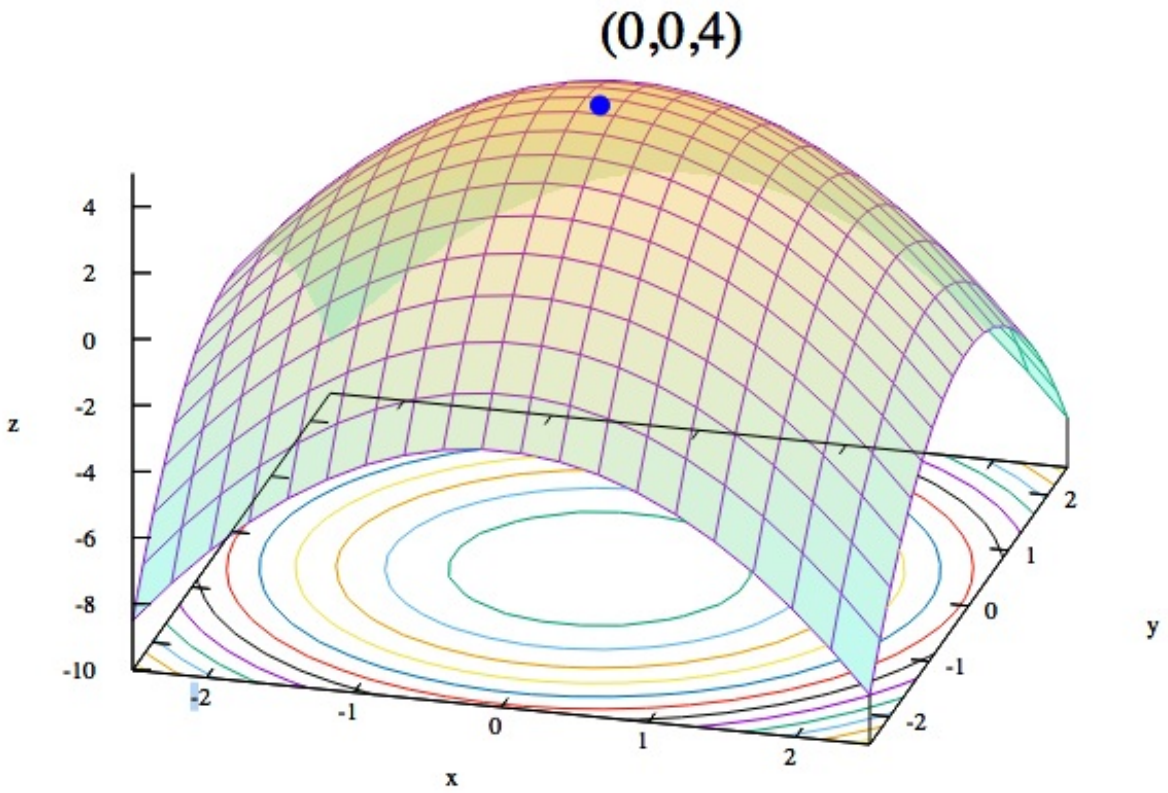
Fitting the Training Data



Optimization

- Iteratively adjust the values of the x parameter vector
  - Until we minimize the error in the model
- Error = prediction – actual
- Loss functions measure error
  - simple/common loss function:
  - “mean squared error”
- How do we make choices about the next iterative “step”?
  - Where “step” is how we change the x parameter vector

Convex Optimization



Gradient Descent

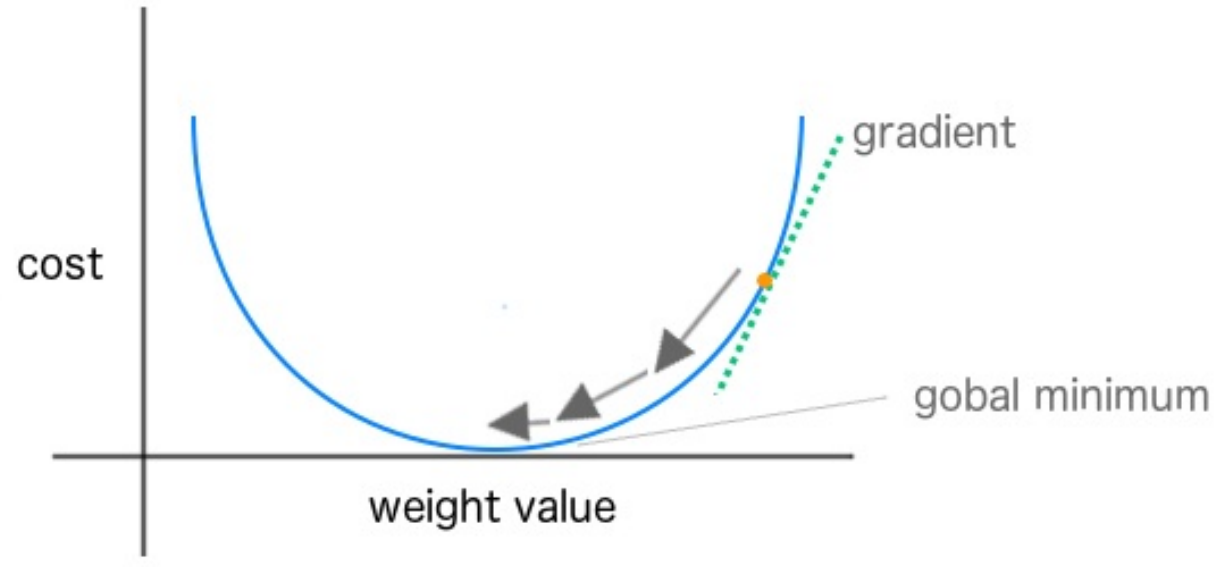
- Optimization method where we consider parameter space as
  - “hills of error”
  - Bottom of the loss curve is the most “accurate” spot for our parameter vector
- We start at one point on the curved error surface
  - Then compute a next step based on local information
- Typically we want to search in a downhill direction
  - So we compute the gradient
    - The derivative of the point in error-space
    - Gives us the slope of the curve

Stochastic Gradient Descent

- With basic Gradient Descent we look at every training instance before computing a “next step”
- With SGD with compute a next step after every training instance
    - Sometimes we’ll do a mini-batch of instances

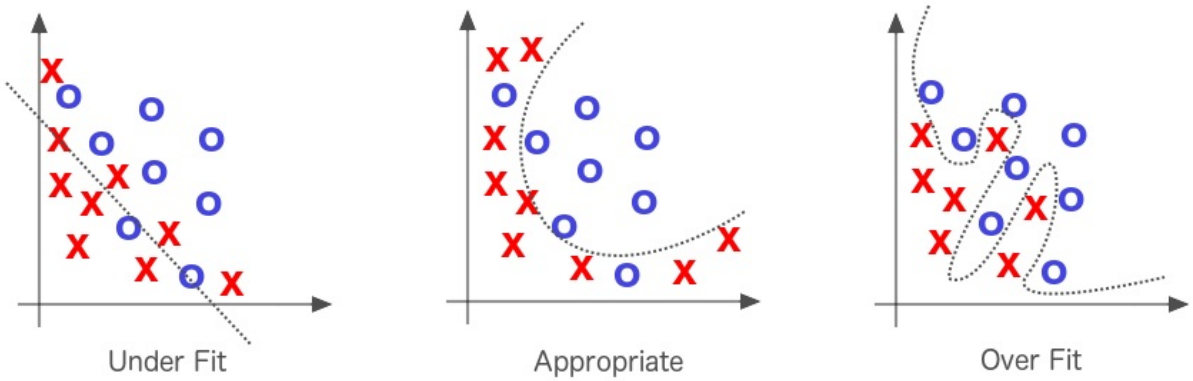
TH-clean this up

SGD Visually Explained



Underfitting and Overfitting

- Underfitting
  - Our model does not learn the structure of the training data well enough
  - Doesn’t perform on new data as well as it could
- Overfitting
  - Our model gives tremendous accuracy scores on training data
  - However, our model performs poorly on test data and other new data



Classification

- A type of answer we can get from a model
- Example:
  - “Is this an image of a cat or a dog?”
  - Binary classification
  - Classes: { cat, dog }
- Binary classification is where we have only 2 labels
  - Example: { positive, negative }
- Multi-Label Classification
  - N number of labels

Supervised vs Unsupervised Learning

- Supervised Learning
  - We give the training process labels (“outputs”) for every training input data row
  - Model learns to associate input data with output value
- Unsupervised Learning
  - No labels
  - Model attempts to learn structure in the data

Regression

- Where we seek a continuous value output from the model
- Example: “predict the temperature for tomorrow”
  - Output: 75F
- Example: “predict price of house based on square footage”
  - Output: $250,000.00

Clustering

- Typically unsupervised learning
  - “K-Means Clustering”
- Example
  - “cluster K groups of similar news articles together”

Logistic Regression

- 3 parts to Logistic Regression Model
  - Hypothesis (logistic function): $\frac{1}{1+e^{-\theta x}}$
    - Gives us a prediction based on the parameter vector x and the input data features
  - Cost Function
    - Example: “max likelihood estimation”
    - Tells us how far off the prediction from the hypothesis is from the actual value
  - Update Function
    - Derivative of the cost function
    - Tells us what direction / how much of step to take [more notes, gradient, etc]

Evaluation and The Confusion Matrix

- Table representing
  - Predictions vs Actual Data
- We count these answers to get
  - True Positives
  - False Positives
  - True Negatives
  - False Negatives
- Allows us to evaluate the model beyond “average accurate” percent
  - Can look at well a model can perform when it needs to be more than just “accurate a lot”

	P’ (Predicted)	N’ (Predicted)
P (Actual)	True Positive	False Negative
N (Actual)	False Positive	True Negative

DL4J OVERVIEW

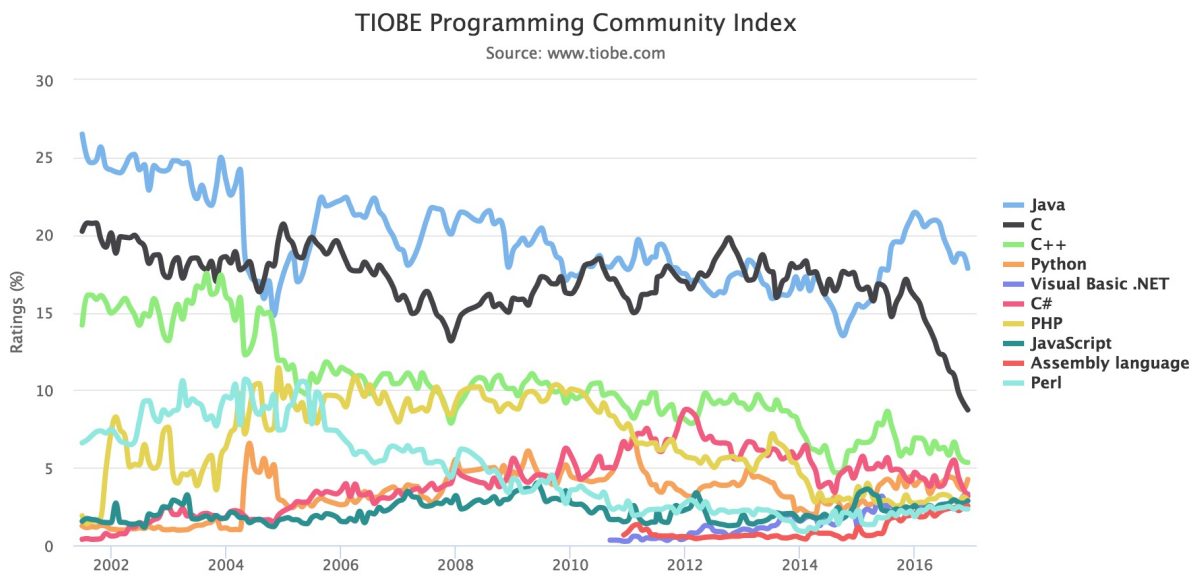
DeepLearning4J Overview

Table of Contents

Goals of the DeepLearning4J project

- Provide a Toolkit for using DeepLearning on the JVM
- Why Java
  - Enterprise users
  - Security
  - Flexibility

Why Java


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
Java
C
C++
Python
Visual Basic .NET
C#
PHP
JavaScript
Assembly language
Perl

DeepLearning4J projects

- DataVec
  - Tools for ETL
- ND4J
  - Numeric Arrays,
  - like Numpy is for Python ND4J is for Java
- libND4J
  - Native Libraries for efficent compute on GPU's/CPU's
- DeepLearning4J
  - Tools to build and train Neural Nets

DataVec

- Neural Nets ingest numeric arrays
- Datavec helps you get from your_data => Numeric Array

DataVec Example

- CSV => NDArray

```
public class CSVExample {

  private static Logger log = LoggerFactory.getLogger(CSVExample.class);

  public static void main(String[] args) throws  Exception {

      //First: get the dataset using the record reader. CSVRecordReader handle
s loading/parsing
      int numLinesToSkip = 0;
      String delimiter = ",";
      RecordReader recordReader = new CSVRecordReader(numLinesToSkip,delimiter
);
      recordReader.initialize(new FileSplit(new ClassPathResource("iris.txt").
getFile()));

      //Second: the RecordReaderDataSetIterator handles conversion to DataSet
objects, ready for use in neural network
      int labelIndex = 4;     //5 values in each row of the iris.txt CSV: 4 in
put features followed by an integer label (class) index. Labels are the 5th va
lue (index 4) in each row
      int numClasses = 3;     //3 classes (types of iris flowers) in the iris
data set. Classes have integer values 0, 1 or 2
      int batchSize = 150;    //Iris data set: 150 examples total. We are load
ing all of them into one DataSet (not recommended for large data sets)

      DataSetIterator iterator = new RecordReaderDataSetIterator(recordReader,
batchSize,labelIndex,numClasses);
      DataSet allData = iterator.next();
```

DataVec Code Explained

- RecordReader recordReader = new CSVRecordReader(numLinesToSkip,delimiter);
  - A RecordReader prepares a list of Writables
  - A Writable is an efficient Serialization format
- DataSetIterator iterator = new RecordReaderDataSetIterator
  - We are in DL4J know, with DataSetIterator
  - Builds an Iterator over the list of records
- DataSet allData = iterator.next();
  - Builds a DataSet

Frequently Used DataVec classes

- CSVRecordReader
  - CSV Text Data
- ImageRecordReader
  - Convert Image to numeric array representing pixel values
- JacksonRecordReader
  - Parses JSON records
- ParentPathLabelGenerator
  - Builds labels based on Directory Path
- Transform, Transform Process Builder, TransformProcess
  - Conversion tools

ND4J

- Provides Scientific Computing Libraries
- Main features
  - Versatile n-dimensional array object
  - Multiplatform functionality including GPUs
  - Linear algebra and signal processing functions

ND4J and DeepLearning

- Neural Nets work with Numerical Arrays
- Classes frequently Used
  - DataSet
  - DataSetIterator

libND4J

- The C++ engine that powers ND4J
  - Speed
  - CPU and GPU support

DeepLEarning4J

- Tools to build and train Neural Networks
- MultiLayerNetworkConfig
  - Build a Neural Network Configuration
- MultiLayerNetwork
  - Intiitalize a Network from a Configuration
- ComputationGraphConfiguration
  - A more flexible Network that MultiLayer
- ComputationGraph
  - Initialize a Computation Graph

DeepLearning4J Frequently used Classes

- MultiLayerNetwork.fit
  - Trains a Model
- Evaluation
  - Evaluates model output against known labelled data
- ModelSerializer
  - Saves and loads trained models
- model.output
  - gets model's output for a single input

DeepLearning4J sample code

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
                .seed(seed)
                .iterations(1)
                .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCEN
T)
                .learningRate(learningRate)
                .updater(Updater.NESTEROVS).momentum(0.9)
                .list()
                .layer(0, new DenseLayer.Builder().nIn(numInputs).nOut(numHiddenNo
des)
                        .weightInit(WeightInit.XAVIER)
                        .activation("relu")
                        .build())
                .layer(1, new OutputLayer.Builder(LossFunction.NEGATIVELOGLIKELIHO
OD)
                        .weightInit(WeightInit.XAVIER)
                        .activation("softmax").weightInit(WeightInit.XAVIER)
                        .nIn(numHiddenNodes).nOut(numOutputs).build())
                .pretrain(false).backprop(true).build();


        MultiLayerNetwork model = new MultiLayerNetwork(conf);
        model.init();
        model.setListeners(new ScoreIterationListener(10));  //Print score every 1
0 parameter updates


        for ( int n = 0; n < nEpochs; n++) {
            model.fit( trainIter );
        }

        System.out.println("Evaluate model....");
        Evaluation eval = new Evaluation(numOutputs);
        while(testIter.hasNext()){
            DataSet t = testIter.next();
            INDArray features = t.getFeatureMatrix();
            INDArray lables = t.getLabels();
            INDArray predicted = model.output(features,false);

            eval.eval(lables, predicted);

        }
```

DEEPLEARNING INTRO

DeepLearning INTRO

This might be too much repeat or need to be merged into other section

Table of Contents

Defining Deep Learning

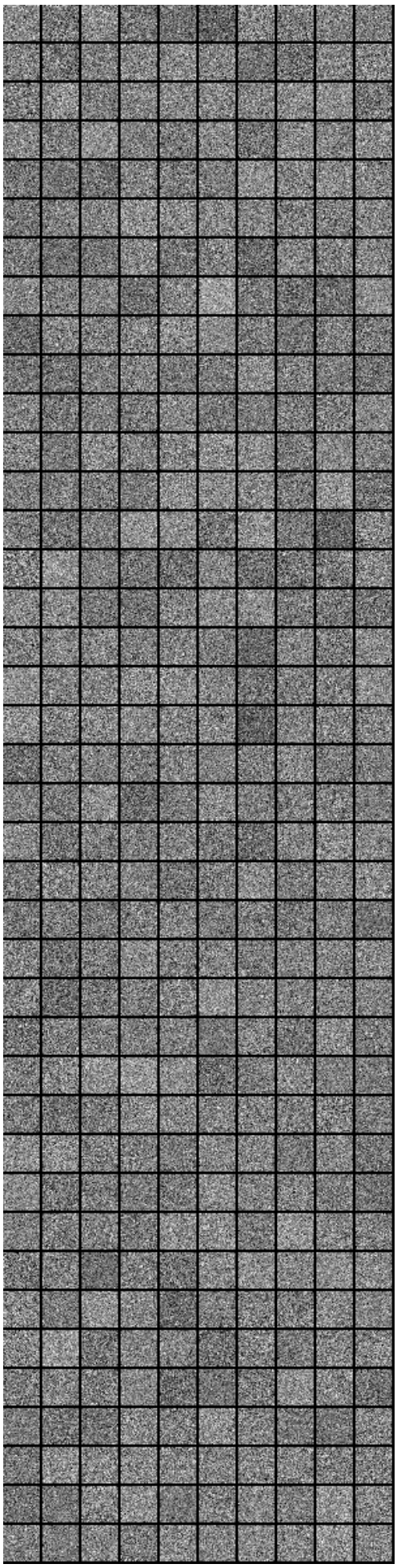
- Higher neuron counts than in previous generation neural networks
- Different and evolved ways to connect layers inside neural networks
- More computing power to train
- Automated Feature Learning

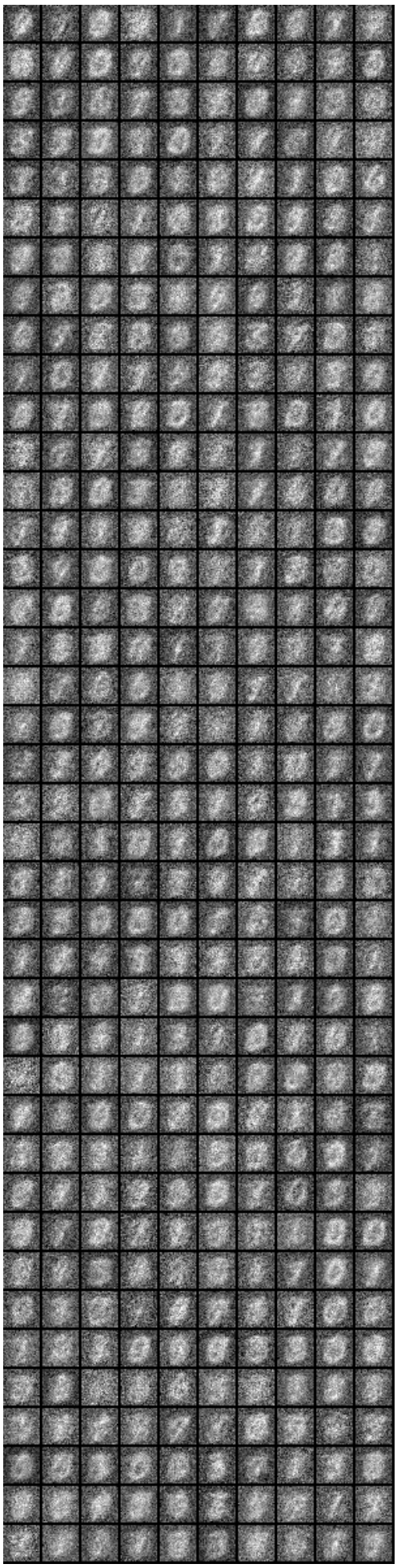
Automated Feature Learning

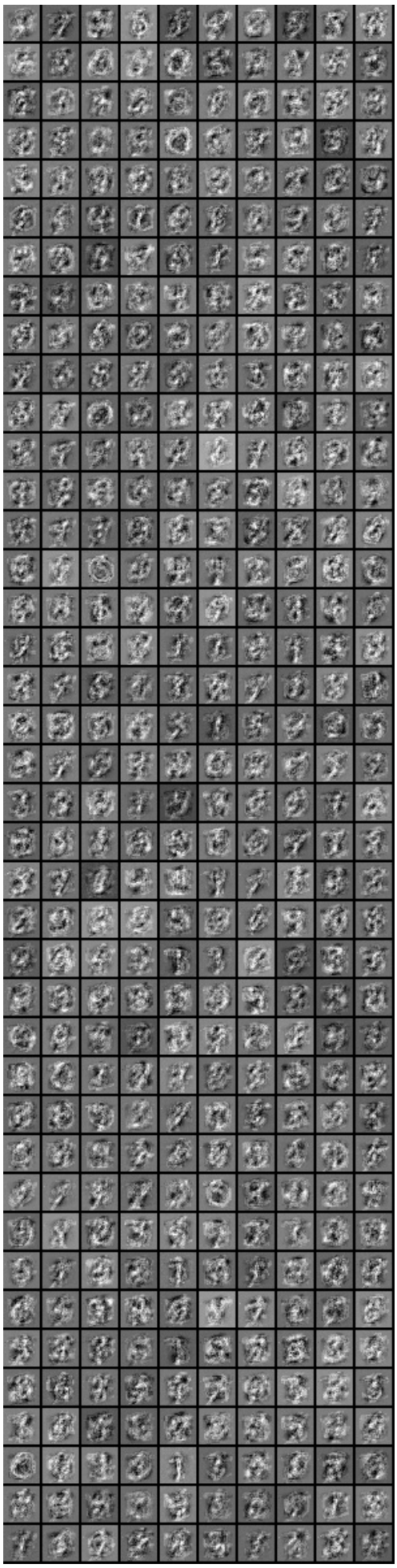
- Deep Learning can be thought of as workflows for automated feature construction
  - From “feature construction” to “feature learning”
- As Yann LeCun says:
  - “machines that learn to represent the world”

MNist Image Learning

- The Following Slides Show a Network as it learns features







Review of Previous Slides

- These are the features learned at each neuron in a Restricted Boltzmann Machine (RBMS)
- These features are passed to higher levels of RBMs to learn more complicated things.

TH- add more stuff on this

Unreasonable Effectiveness: Benchmark Records

1. Text-to-speech synthesis (Fan et al., Microsoft, Interspeech 2014)
2. Language identification (Gonzalez-Dominguez et al., Google, Interspeech 2014)
3. Large vocabulary speech recognition (Sak et al., Google, Interspeech 2014)
4. Prosody contour prediction (Fernandez et al., IBM, Interspeech 2014)
5. Medium vocabulary speech recognition (Geiger et al., Interspeech 2014)
6. English to French translation (Sutskever et al., Google, NIPS 2014)
7. Audio onset detection (Marchi et al., ICASSP 2014)
8. Social signal classification (Brueckner & Schulter, ICASSP 2014)
9. Arabic handwriting recognition (Bluche et al., DAS 2014)
10. TIMIT phoneme recognition (Graves et al., ICASSP 2013)
11. Optical character recognition (Breuel et al., ICDAR 2013)
12. Image caption generation (Vinyals et al., Google, 2014)
13. Video to textual description (Donahue et al., 2014)
14. Syntactic parsing for Natural Language Processing (Vinyals et al., Google, 2014)
15. Photo-real talking heads (Soong and Wang, Microsoft, 2014).

Four Major Architectures

- Deep Belief Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
- Recursive Neural Networks

The More Things Change…

- Deep Learning is still trying to answer the same fundamental questions such as:
  - “is this image a face?”
- The difference is Deep Learning makes hard questions easier to answer with better architectures and more computing power
  - We do this by matching the correct architecture w the right problem

Quick Usage Guide

- Timeseries or Audio Input
  - Use a Recurrent Neural Network
  - Examples: Fraud Detection, Anomaly Detection
- Image input
  - Use a Convolutional Neural Network
- Video input
  - Use a hybrid Convolutional + Recurrent Architecture!

Common Architectural Principals

- Layer-oriented architecture
  - But have different types of hidden layers

```
* Different schemes of connectivity
```

  - Connection weights are still parameters
- Activation functions control how information propagates from one layer to next
- Input / Output layer concepts still the same

Evolution of Layers

- Layers evolve to have different types of connections
- Classic FFNN: Fully-Connected
- Convolutional Neural Network
  - Connected to spatially-local areas of previous layer
  - Connected to full depth of output
- Recurrent Neural Networks
  - Connections from previous timesteps

Activation Functions

- Sigmoid is considered a classical neural network activation function
  - Fallen out of favor more recently
- Model Deep Networks use
  - Rectified Linear Units (ReLU)
  - TanH

Activation Functions Illustrated

Add pictures of each, describe Vanishing Gradient Problem or not

Loss Functions

Need good definition here

- Regression
  - Squared Loss
- Classification
  - Hinge Loss
    - Binary classifier (“hard classification”)
- Logistic Loss
  - When we want probabilities as opposed to hard classifications

Optimization Algorithms

- Stochastic Gradient Descent
  - Most common
- L-BFGS
- Conjugate Gradient
- Hessian Free

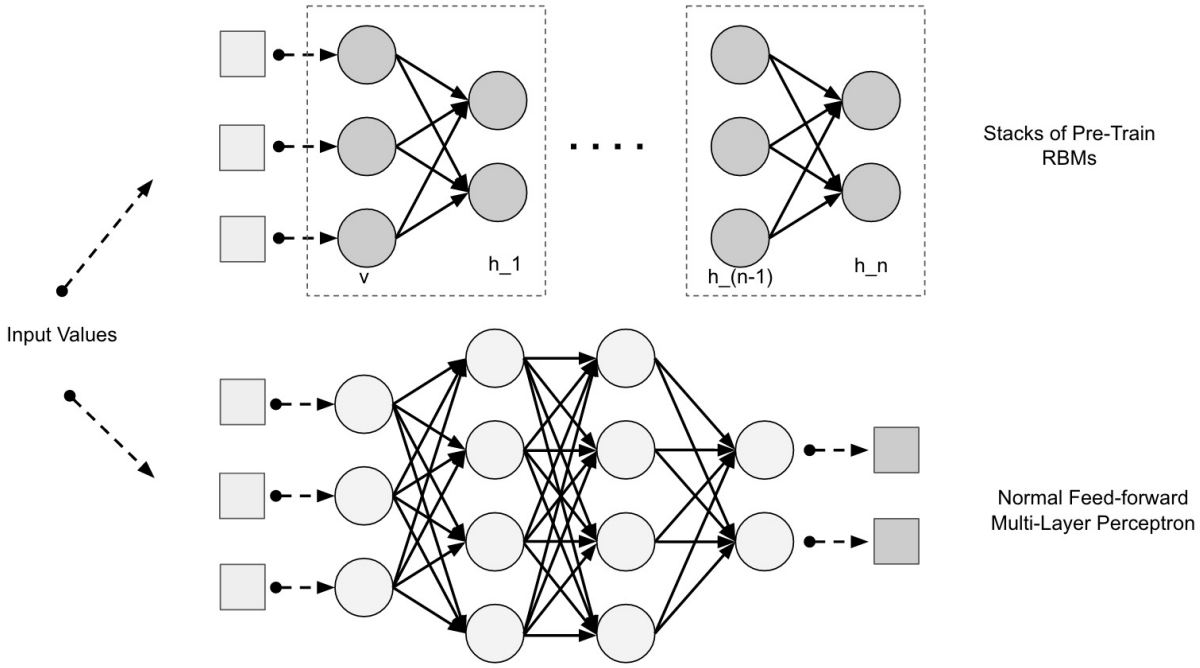
Hyperparameters

- Layer Count
- Learning Rate
- Parameter Count
  - Neurons per layer
- Loss Function Type
- Optimization Algorithm

Building Blocks of Deep Networks

- Some networks are composed of other networks
  - Use sub networks to extract features
- Example Deep Belief Networks
  - Use a set of Restricted Belief Networks to extract good initial parameter values
- AutoEncoders
  - Can learn to find a minimal representation of the input data

Deep Belief Network



DEEPLEARNING INTRO

Recurrent Neural Networks

Table of Contents

Recurrent neural networks

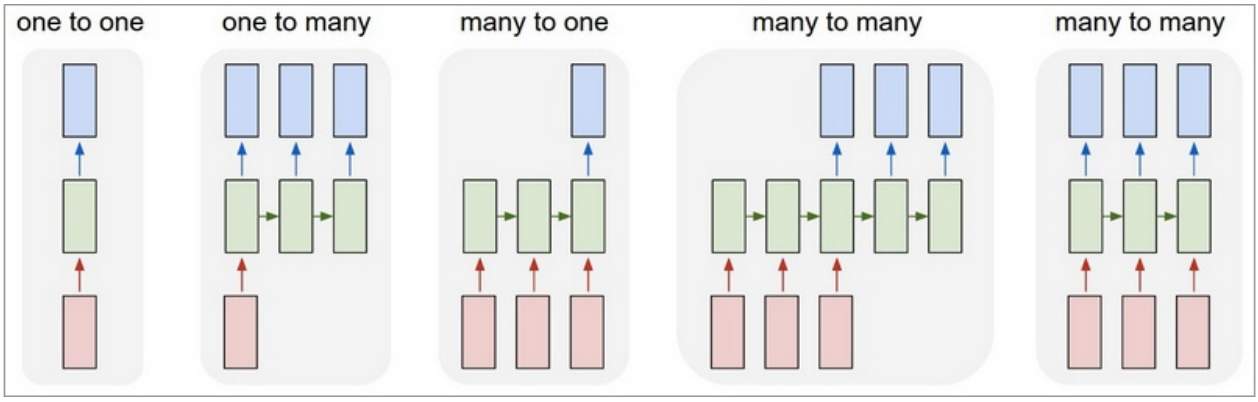
Recurrent neural networks

- Family of feedforward networks
  - Differ in how they send information over timesteps
- Allows for modeling change in vectors over time
  - Multiple sets of vectors as inputs
  - As opposed to a single input feature vector

Timeseries and Recurrent Networks

- When dealing with sequential or timeseries data
  - We prefer to apply Recurrent Networks
- Allows us to plug in how the data changes over time
  - Patient data collected periodically
  - State of power grid over time
  - Sequence of actions by customer

RNN Architectures



Add Captions somehow, or rebuild image

- Standard supervised learning
- Image Captioning
- Sentiment Analysis
- Video Captioning, Natural Language Translation
- Part of Speech Tagging
- Generative Mode for text

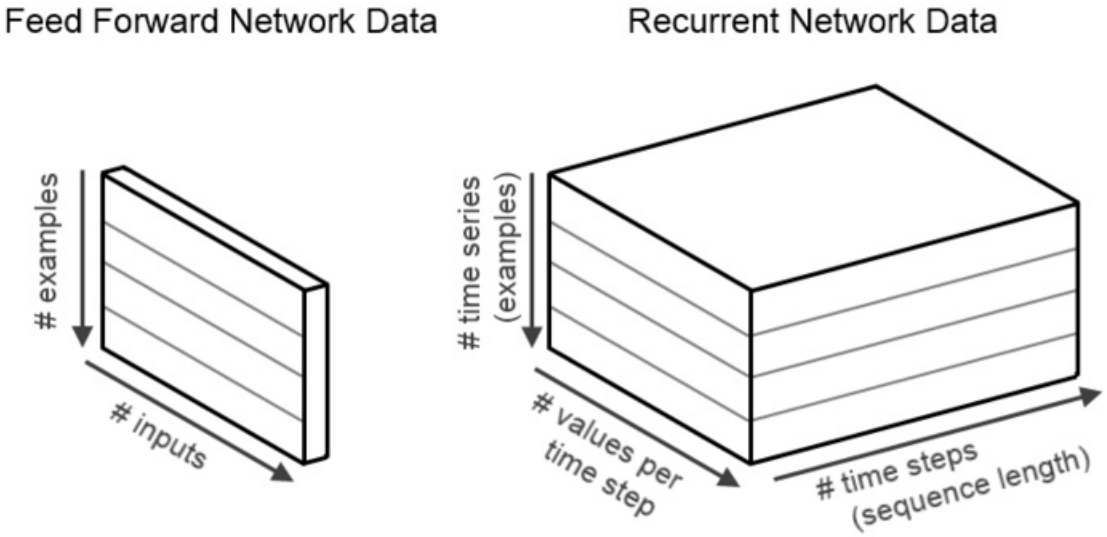
Example: PhysioNet Raw Data

- Set-a
  - Directory of single files
  - One file per patient
  - 48 hours of ICU data
- Format
  - Header Line
  - 6 Descriptor Values at 00:00
  - Collected at Admission
  - 37 Irregularly sampled columns
  - Over 48 hours

Physionet Data

Time,Parameter,Value 00:00,RecordID,132601 00:00,Age,74 00:00,Gender,1 00:00,Height,177.8 00:00,ICUType,2 00:00,Weight,75.9 00:15,pH,7.39 00:15,PaCO2,39 00:15,PaO2,137 00:56,pH,7.39 00:56,PaCO2,37 00:56,PaO2,222 01:26,Urine,250 01:26,Urine,635 01:31,DiasABP,70 01:31,FiO2,1 01:31,HR,103 01:31,MAP,94 01:31,MechVent,1 01:31,SysABP,154 01:34,HCT,24.9 01:34,Platelets,115 01:34,WBC,16.4 01:41,DiasABP,52 01:41,HR,102 01:41,MAP,65 01:41,SysABP,95 01:56,DiasABP,64 01:56,GCS,3 01:56,HR,104 01:56,MAP,85 01:56,SysABP,132 …

Preparing Input Data

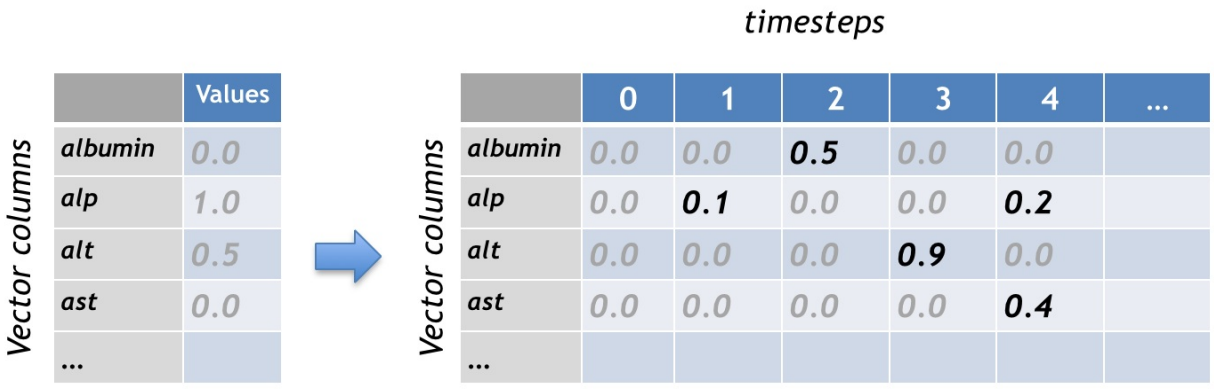


- Input was 3D Tensor (3d Matrix)
  - Mini-batch as first dimension
  - Feature Columns as second dimension
  - Timesteps as third dimension
- PhysioNet: Mini-batch size of 20, 43 columns, and 202 Timesteps
  - We have 173,720 values per Tensor input

TH- explain batch and minibatch in terms of training

Input Sequence

- A single training example gets the added dimension of timesteps per column



Uneven Timesteps and Masking

Single Input

(columns + timesteps)

	0	1	2	3	4	...
albumin	*0.0*	*0.0*	***0.5***	*0.0*	*0.0*	
alp	*0.0*	***0.1***	*0.0*	*0.0*	*0.0*	
alt	*0.0*	*0.0*	*0.0*	***0.9***	*0.0*	
ast	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	
...						

Input Mask

(only timesteps)

1.0	1.0	1.0	1.0	0.0	0.0

Recurrent Networks For Classification

- This is the “many-to-one” setup
  - Traditionally we’d do hand coded feature extraction on timeseries and encode into a vector
    - Losing the time aspect to the data
  - The “many”-part allows us to input a sequence without losing the time domain aspect
- Input is a series of measurements aligned by timestep
  - 0,1,0,0
  - 1,0,1,1
- Output in this case is a classification
  - Example: “Fraud vs Normal transaction”

Sequence Classification with RNNs

- Recurrent Neural Networks have the ability to *model change of input over time*
- Older techniques (mostly) do not retain time domain
  - Hidden Markov Models do…
    - *but are more limited*
- Key Takeaway:
  - For working with Timeseries data, RNNs will be more accurate

DEEPLEARNING INTRO

ETL and Vectorization

Table of Contents

Concepts in ETL: DataVec

What is ETL?

- Extract
  - Pull data from a source

```
* Logs
* Another Database
```

- Transform
  - Convert each column with a function
  - Filter some columns out
- Load
  - Create a new dataset / table
  - Setup to be used by another application

What is Vectorization?

- Convert each column in a table/dataset into a floating point number
- Four Attribute Types
  - Nominal
  - Ordinal
  - Interval
  - Ratio
- Raw Text has many complications
  - Bag of Words / Counts
  - TF-IDF
- Sometimes we need to enumerate the options for the column
  - Option A -> 0.0, Option B -> 1.0, …

Examples of Raw Data Sources

- Raw text documents
- A file containing a text record per line
- Binary timeseries data in custom file format
- Pre-processed datasets with a mixture of numeric and string attributes
- Image Files
- Audio Files
- Video Files

What is DataVec?

- Library for handling machine learning data / ETL
  - (Extract, Transform, Load)
- Goal is to simplify the preparation and loading of raw data into a format ready for use for machine learning
  - Also provides vectorization functionality
- DataVec includes functionality for loading
  - Tabular (CSV, etc)
  - Image
  - Time series

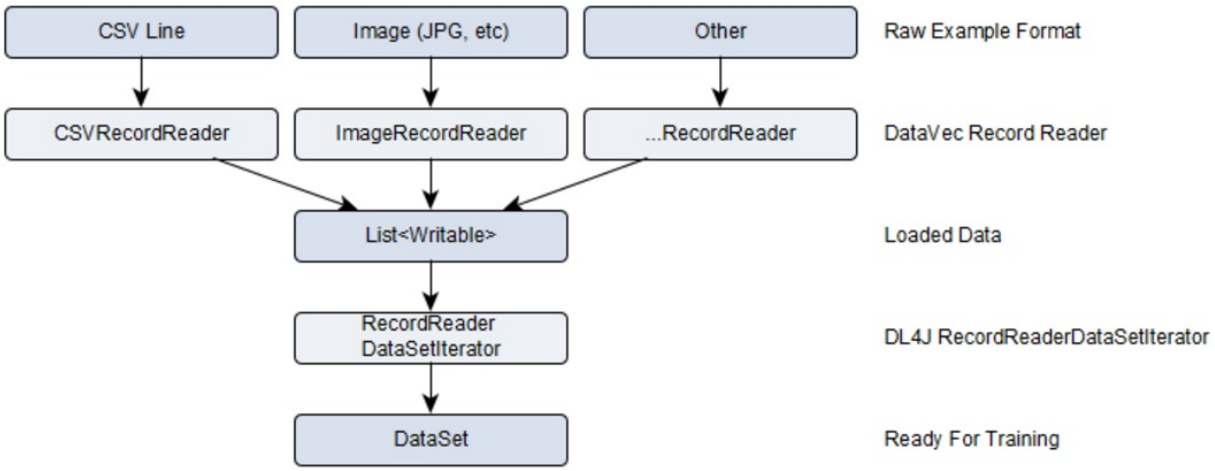
Major Facets of DataVec

- Mode of Execution
  - Local
  - Spark
- Type of Data
  - Tabular (“database table”)
  - Sequential (“Timeseries”)

DataVec Abstractions

- Writable
  - Interface representing a piece of data
  - Example: DoubleWritable
- RecordReader
  - Interface to provide mechanism to load data from raw file format
  - Converts data to List
- RecordReaderDataSetIterator
  - Conversion of List to DataSet

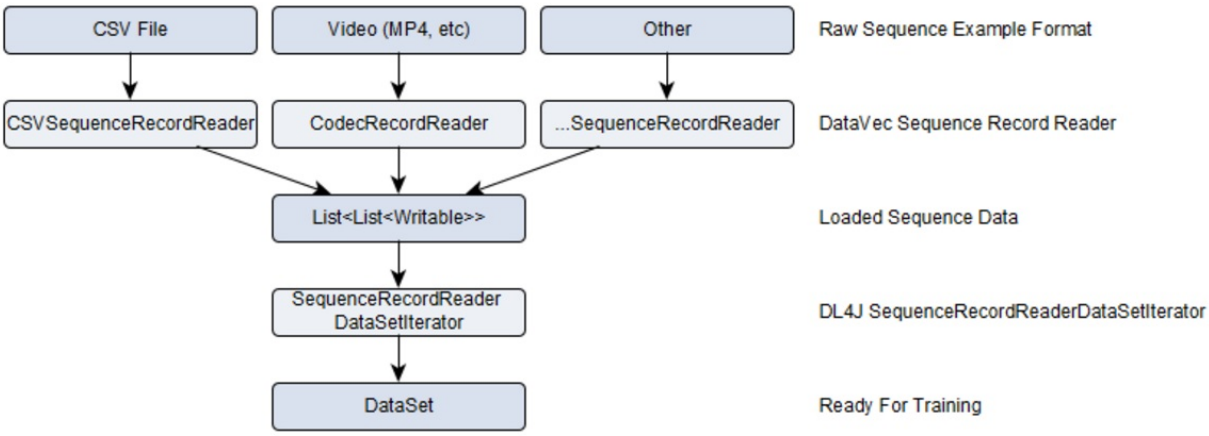
DataVec Processing Pipelines



DataVec Handling Sequences

- More complex data type
  - More complicated to store / load
- Move from List (a vector)
  - To a series of vectors represented by:
    - List
  - Represents the input of a single sequence
- More complex input, but still trying to produce a DataSet
  - Input is now a Tensor!

DataVec Sequence Processing Pipelines



Sequences and Local datavec pipelines

Goal: DataSet from Local Raw Data

- Need to be able to load the raw sequence data
  - Read the file format
  - Match up labels to features
  - Perform any normalization during vectorization
- If the input file matches a record reader
  - We can leverage that code to build our NDArrays and DataSets
- If not, we need to write custom code
  - to build the DataSet objects

Example Data Layout

- Suitable for CSVSequenceRecordReader for sequence input
  - One timeseries per file
  - Separate file for labels
- Example
  - Features for series 0:
    - Train/features/0.csv
  - Label for series 0:
    - Train/labels/0.csv
- This dataset is univariate (one column in CSV file)

High-Level Pattern

- Load data from File:
  - CSVSequenceRecordReader
- Dealing with file format:
  - NumberedFileInputSplit
- We handle raw sequence data to DataSet (tensor) conversion with:
  - SequenceRecordReaderDataSetIterator

Local Code Example

```
//Note that we have 450 training files for features: train/features/0.csv through
train/features/449.csv
SequenceRecordReader trainFeatures = new CSVSequenceRecordReader();
trainFeatures.initialize(new NumberedFileInputSplit(featuresDirTrain.getAbsolutePa
th() + "/%d.csv", 0, 449));
SequenceRecordReader trainLabels = new CSVSequenceRecordReader();
trainLabels.initialize(new NumberedFileInputSplit(labelsDirTrain.getAbsolutePath()
 + "/%d.csv", 0, 449));

int miniBatchSize = 10;
int numLabelClasses = 6;
DataSetIterator trainData = new SequenceRecordReaderDataSetIterator(trainFeatures,
 trainLabels, miniBatchSize, numLabelClasses,
    false, SequenceRecordReaderDataSetIterator.AlignmentMode.ALIGN_END);

//Normalize the training data
DataNormalization normalizer = new NormalizerStandardize();
normalizer.fit(trainData);              //Collect training data statistics
trainData.reset();

//Use previously collected statistics to normalize on-the-fly. Each DataSet return
ed by 'trainData' iterator will be normalized
trainData.setPreProcessor(normalizer);
```

Reading Data From File

- We need a record reader
  - Start with: SequenceRecordReader
  - Initialize with a specific subclass of input split
    - This deals with the specific file format
- Supported data formats
  - CSVSequenceRecordReader
    - Reads CSV based data from CSV Text Files
  - CSVNLinesSequenceRecordReader
    - version of CSV sequence reader: each sequence is exactly N lines long, all in one file, one after the other
  - RegexSequenceRecordReader
    - Good for log data
  - CollectionSequenceRecordReader
    - Mainly for debugging/testing

Dealing w File Formats

- Supported File Formats
  - CSV Text Files
    - Numbered Files: NumberedFileInputSplit
      - Ex: numbered files in directory -> NumberedFileInputSplit
- Notes
  - Any input split with URIs: FileSplit for example. Totally fine for "one sequence per file" cases
    - BUT: be careful of alignment for features and labels in separate files. Something like NumberedFileInputSplit is better for this

Building a DataSet Iterator

- Why?
  - Need a place that takes raw data from possibly multiple files + label data and places it all in a single DataSet object
- Base Class:
  - DataSetIterator
- Sequence Specific Class:
  - SequenceRecordReaderDataSetIterator

Handling Normalization

```
DataSetIterator trainData = …

DataNormalization normalizer = new NormalizerStandardize();

// collect statistics
normalizer.fit(trainData);
trainData.reset();

// use statistics to vectorize data
trainData.setPreProcessor(normalizer);
```

Vectorization Techniques for Normalization

Techniques of Normalization

- When we normalize at the vector level
  - most of the time this ends up being a division of the vector by a norm (in this case, “length”) of the vector.
- Standardize
  - ZMUV
- Min-Max Scaling (“feature scaling”)
- Binarization

Standardization

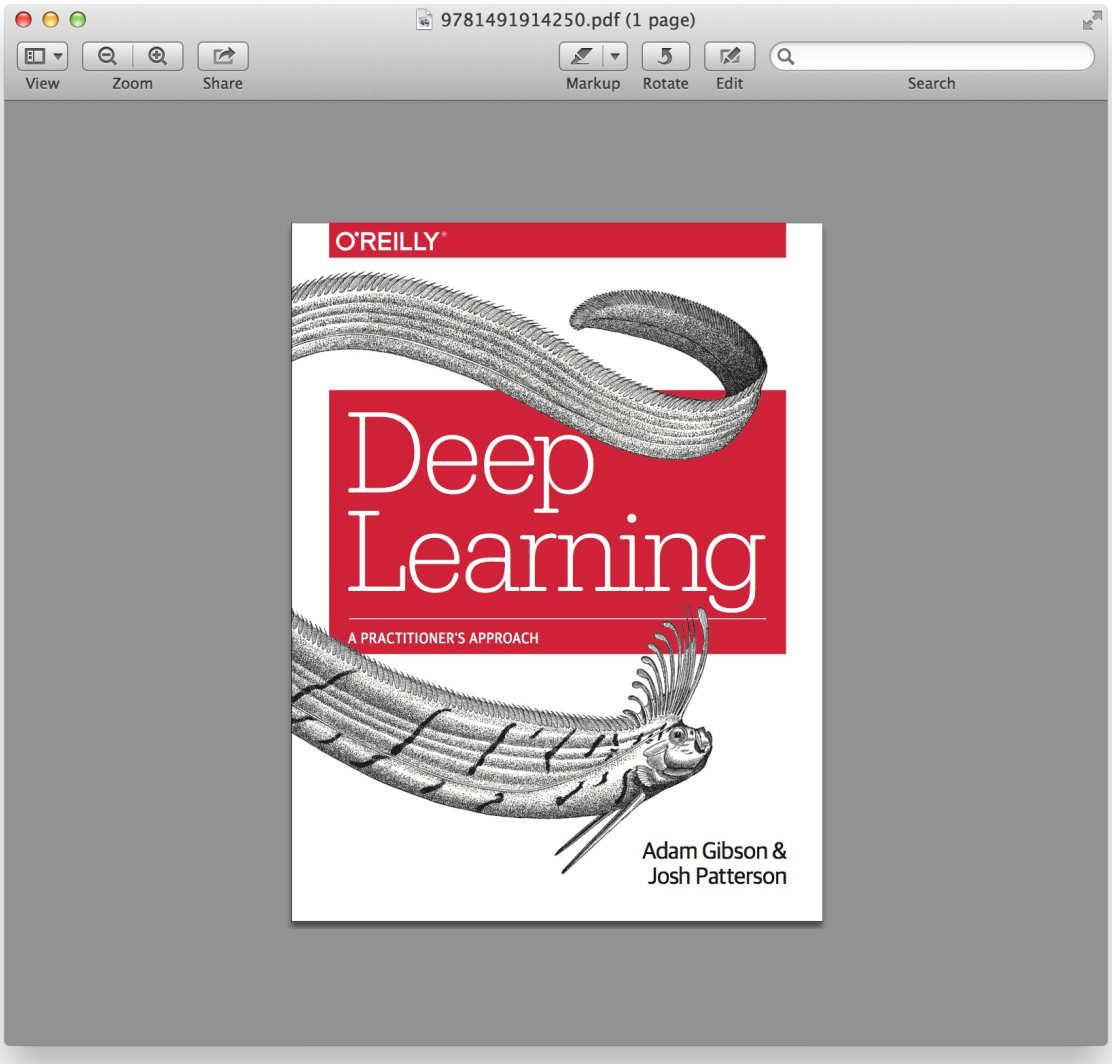
- When we “standardize” a vector we subtract a measure of location (minimum, maximum, median, etc)
  - and then divide by a measure of scale (variance, standard deviation, range, etc).
  - Zero Mean, Unit Variance

TH- show example Use degrees celsius vs farhenheit

Binarization

- Use filter to produce a feature that includes a 1 or a 0 as its value
- We set threshold on filter function as the threshold
- Where do we use it?
- [todo]

Want to know more?



DEEPLEARNING INTRO

Building recurrent neural network applications

Table of Contents

Building recurrent neural network applications

Overview

- Basics of Building RNN Input
- Building Input for Local Training
- Building Input for Spark Training
- Model Architecture, Training, and Evaluation

Goal: Model Sensor Data with RNN

Where is data?

- What format is data in?
  - Leverage ideas from Wednesday's last unit on ETL!
  - Convert raw data to DataSet
- Setup a RNN Model
  - Architecture
- Executing training
  - Local
  - Spark

Basics of RNN Input

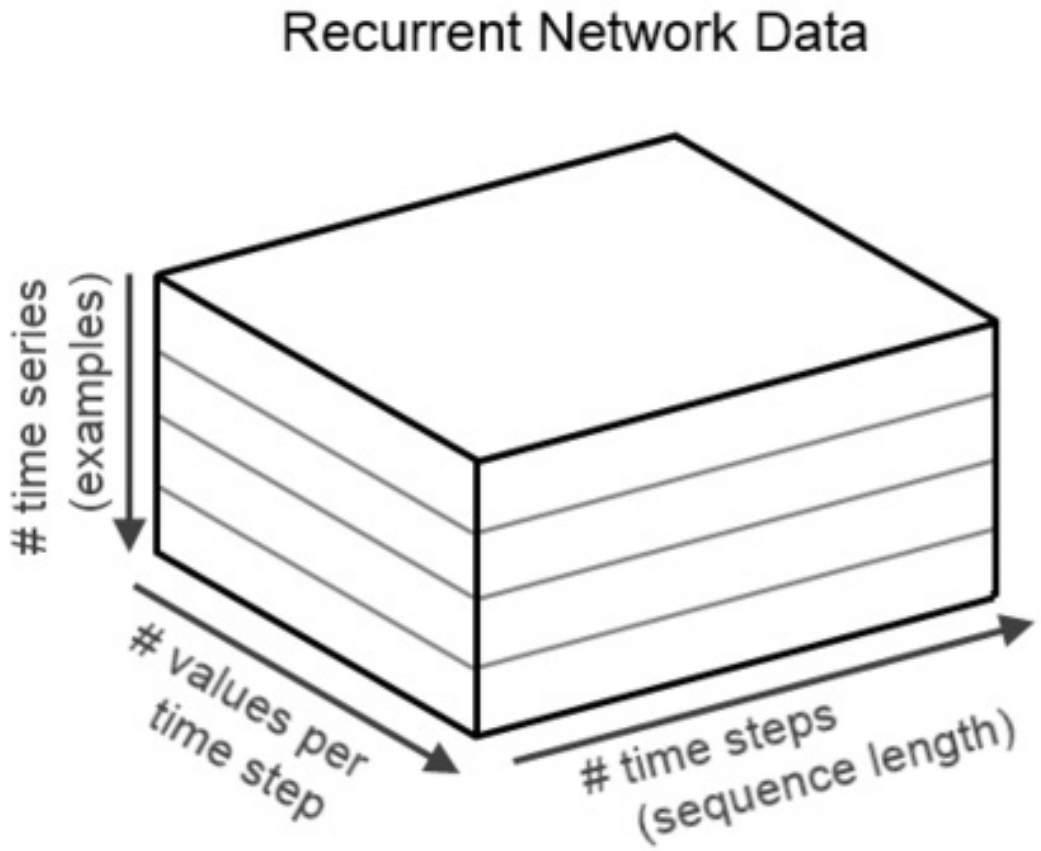
From Raw Input to NDArrays

- Raw Input

0	1	2	3	4	…
albumin	0.0	0.0	0.5	0.0	0.0
alp	0.0	0.1	0.0	0.0	0.2
alt	0.0	0.0	0.0	0.9	0.0
ast	0.0	0.0	0.0	0.0	0.4

From Raw Input to NDArrays

- NDArray



Question

- Given a data set with two values
  - Temperature taken 1 per hour
  - Total precipitation taken once a day at hour 24
- Describe the data format
  - How many data points in a days worth of data
  - What if temp was taken every minute?

Key DataSet Vectorization Questions

- How is the input data laid out?
  - File format?
  - Single or Multiple Files?
  - How many columns per sensor?
  - Where are the labels located?

File Format

- Generally text input format
- If not text, may need a custom input format

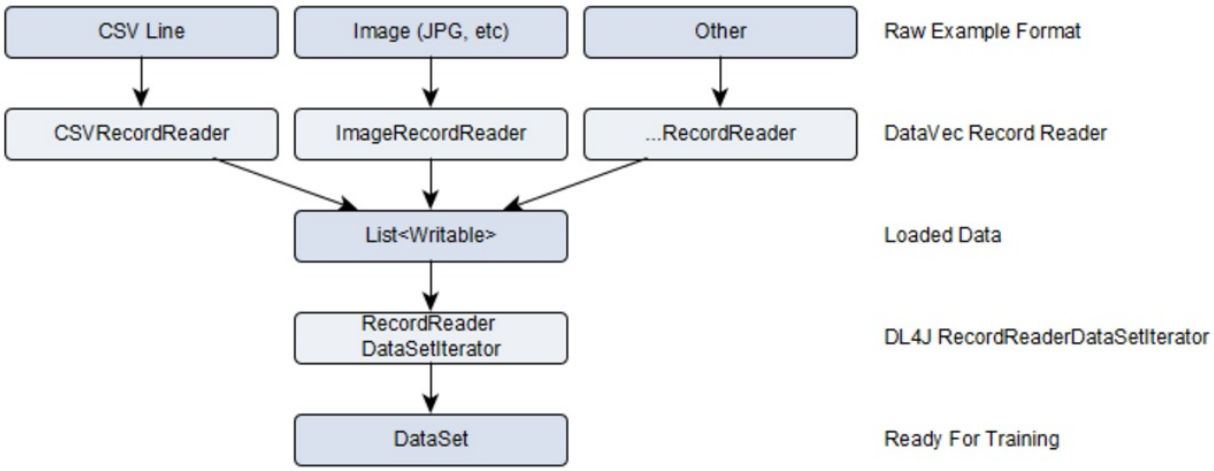
Single or Multiple Files?

- Timeseries readings in a single file
  - “each line is a CSV set of readings from a single sensor”
  - “each line is a sentence”
- Timeseries readings in multiple files
  - “each file represents readings from a single sensor”

Columns per Sensor

- How many different types of readings are being taken per source?
  - Example of single: “voltage”
    - 1 column
  - Example of multiple: “voltage” and “temperature”
    - 2 columns

Preparing Input Data: Tensors



- Input was 3D Tensor (3d Matrix)
  - Mini-batch as first dimension
  - Feature Columns as second dimension
  - Timesteps as third dimension
- Goal: produce a DataSet object
  - Containing vectorized input data

A Single Training Example

A single training example gets the added dimension of timesteps per column

	Values
albumin	0.0
alp	1.0
alt	0.5
ast	0.0
...	

TimeSteps

	0	1	2	3	4	....
albumin	0.0	0.0	0.5	0.0	0.0	...
alp	0.0	0.1	0.0	0.0	0.2	...
alt	0.0	0.0	0.0	0.9	0.0	...
ast	0.0	0.0	0.0	0.0	0.4	...
...	...	...	...	...	...	...

Creating RNN DataSets

- D = new DataSet(input, labels, mask, mask2)
- Input
  - INDArray input = Nd4j.zeros(new int[]{ miniBatchSize, INPUTColumnCount, timesteps });
- Labels
  - INDArray labels = Nd4j.zeros(new int[]{ miniBatchSize, OUTPUTColumnCount, timesteps });
- Mask
  - Nd4j.zeros(new int[]{ miniBatchSize, timesteps });
- Mask2
  - (Same as mask)

Masking

- Masks used for training data and labels
  - We set the mask timestep value to 1.0 for every timestep containing training data
  - We set the mask timestep value to 0.0 for all other timesteps

High-Level Code: Building DataSets

- Typically we want to loop across the timesteps in the outer loop
- For each timestep
  - Set the relevant columns to the values
  - INDArray.setScalar(col, timestep, value)
- This will loop through input vectorized data
  - Merge it into a row in a timestep

Building Tensors with NDArrays

- When setting values for a single record
  - INDArray.putScalar(vectorColumn, timestepIndex, value)
- When setting values for a record in a mini-batch
  - INDArray.putScalar(miniBatchIndex, vectorColumn, timestepIndex, value)

Building Sequential Tensor input for

Local Training

Goal: DataSet from Local Raw Data

- Need to be able to load the raw data
  - Read the file format
  - Match up labels to features
  - Perform any normalization during vectorization
- If the input file matches a record reader
  - We can leverage that code to build our NDArrays and DataSets
- If not, we need to write custom code
  - to build the DataSet objects

Local Code Example

```
//Note that we have 450 training files for features: train/features/0.csv through
train/features/449.csv
SequenceRecordReader trainFeatures = new CSVSequenceRecordReader();
trainFeatures.initialize(new NumberedFileInputSplit(featuresDirTrain.getAbsolutePa
th() + "/%d.csv", 0, 449));
SequenceRecordReader trainLabels = new CSVSequenceRecordReader();
trainLabels.initialize(new NumberedFileInputSplit(labelsDirTrain.getAbsolutePath()
 + "/%d.csv", 0, 449));

int miniBatchSize = 10;
int numLabelClasses = 6;
DataSetIterator trainData = new SequenceRecordReaderDataSetIterator(trainFeatures,
 trainLabels, miniBatchSize, numLabelClasses,
    false, SequenceRecordReaderDataSetIterator.AlignmentMode.ALIGN_END);

//Normalize the training data
DataNormalization normalizer = new NormalizerStandardize();
normalizer.fit(trainData);              //Collect training data statistics
trainData.reset();

//Use previously collected statistics to normalize on-the-fly. Each DataSet return
ed by 'trainData' iterator will be normalized
trainData.setPreProcessor(normalizer);
```

High-Level Pattern

- Load data from File:
  - **SequenceRecordReader** subclass
- Dealing with file format:
  - **InputSplit** subclass
- We handle raw sequence data to DataSet (tensor) conversion with:
  - **SequenceRecordReaderDataSetIterator**
  - Also works with **DataNormalization** object

Building Tensor Input for

Spark Training

Representing Timeseries with Spark

- First goal: Load data from Disk
  - RDD>
    - can't represent sequence data...
  - RDD>>
    - Allows us to represent collections of timeseries
- Second Goal: convert to DataSet
  - RDD>> becomes
    - RDD

High-Level Pattern: Spark

- Load data from File:
  - Use the spark context to load the raw data into RDDs
  - Have to consider loading data from a single file vs multiple files in choosing methods
  - Produces a RDD or RDD
- Dealing with file format:
  - Assuming HDFS as storage
    - Use Hadoop input formats with Spark
- We handle raw sequence data to DataSet (tensor) conversion with:
  - Custom class:
    - implements Function

Load Data From HDFS

- Use Input formats from Hadoop
  - CSV Files: TextInputFormat (default)
- Loading from a single file
  - JavaRDD trainingRecordCSVLines = sc.textFile(pathData);
- Loading Data from multiple files
  - JavaPairRDD linesTR = sc.wholeTextFiles(…)

Writing Custom Timeseries to DataSet Converter

- Implements method:
  - public DataSet call(String s) throws Exception {
- Creates 2D dataset for Features (NDArray)
  - Columns x timesteps
  - (Input)
- Create labels NDArray
  - (Output)
- Map both NDArrays together in a DataSet object
  - { Input -> Output }

Custom Converter Code Example

```
@Override
public DataSet call(String s) throws Exception {

    recordReader.initialize(new StringSplit(s));
    List<Writable> lw = recordReader.next();

    int inputColumnCount = 1;
    int outputColumnCount = this.numOutComes;
    int maxTimestepLength = lw.size() - 1;

    INDArray input  = Nd4j.zeros(new int[]{ 1, inputColumnCount, maxTimestepLength
 });
    INDArray labels = Nd4j.zeros(new int[]{ 1, outputColumnCount, maxTimestepLengt
h });
    INDArray mask   = Nd4j.zeros(new int[]{ 1, maxTimestepLength });
    INDArray mask2 = Nd4j.zeros(new int[]{ 1, maxTimestepLength });

    int csvIndex = 0;

    // TODO: probably should account for a label index other that index 0...
    for ( int timestepIndex = 0; timestepIndex < maxTimestepLength; timestepIndex+
+ ) {

        // set the features
        input.putScalar( new int[]{ 0, 0, timestepIndex }, lw.get( csvIndex + 1 ).
toDouble() );

        // set the mask
        mask.putScalar(new int[]{ 0, timestepIndex }, 1.0);

        // set the label for every timestep at the class column
        labels.putScalar(new int[]{ 0, lw.get( 0 ).toInt() - 1, timestepIndex }, 1
.0);

        csvIndex++;


    }


    Nd4j.copy(mask, mask2);

    return new DataSet(input,labels, mask, mask2);
}
```

Code: Converting RDD>>

```
// convert raw data into sequence
JavaRDD<List<List<Writable>>> timeseries_ETLd = executor.executeToSequence(parsed,
 tp);

…

// data cleanup transform (optional)
JavaRDD<List<List<Writable>>> vectorizedTimeseriesResult = executor.executeSequenc
eToSequence(timeseries_ETLd, tpVectorize);

...

// now convert to DataSet for modeling
JavaRDD<DataSet> timeseriesDataSet = vectorizedTimeseriesResult.map(
new DataVecSequenceDataSetFunction( 0, 2, false, null, null ) );
```

Spark Timeseries Normalization

- NormalizerStandardize and NormalizerMinMaxScaler handle time series
  - e.g., mean/stdev etc are over all time steps, not independent for each, etc

Rnn Model Architecture, training, and evaluation

Common RNN Model Architecture

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
                .seed(123)
                .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCEN
T).iterations(1)
                .weightInit(WeightInit.XAVIER)
                .updater(Updater.NESTEROVS).momentum(0.9)
                .learningRate(0.005)
                .gradientNormalization(GradientNormalization.ClipElementWiseAbsolu
teValue)
                .gradientNormalizationThreshold(0.5)
                .list()
                .layer(0, new GravesLSTM.Builder().activation("tanh").nIn(1).nOut(
10).build())
                .layer(1, new RnnOutputLayer.Builder(LossFunctions.LossFunction.MC
XENT)
                        .activation("softmax").nIn(10).nOut(numLabelClasses).build
())
                .pretrain(false).backprop(true).build();

        MultiLayerNetwork net = new MultiLayerNetwork(conf);
        net.init();
```

Model Training and Evaluation

```
int nEpochs = 40;
String str = "Test set evaluation at epoch %d: Accuracy = %.2f, F1 = %.2f";
for (int i = 0; i < nEpochs; i++) {
    net.fit(trainData);
    Evaluation evaluation = net.evaluate(testData);
    log.info(String.format(str, i, evaluation.accuracy(), evaluation.f1()));
    testData.reset();
    trainData.reset();
}
log.info("----- Example Complete -----");
```