

Welcome to the Course



This course is a Hands-ON 3 Day Course

- Topics Covered
 - Machine Learning
 - Deep Learning
 - Building Neural Networks with DeepLearning4j
 - ETL processes with DataVec



Schedule

- 9:00 AM Start
- 12:00 PM -> 1:00 PM Lunch
- Breaks 10 minutes each hour



Labs

- Java lab exercises using IntelliJ
- Separate Lab Document
 - Simplest Network Lab
 - DataVec Lab
 - FeedForward Network Lab
 - Convolutional Neural Network Lab
 - RNN/LSTM Lab
 - Model Saving Lab



Contents

- Introduction
- What is Deep Learning
- Neural Network Demonstration
- Types of Neural Networks
- DeepLearning4J
- DeepLearning4J Overview
 - DataVec
 - DataVec Lab
 - ND4J and libnd4j
 - DeepLearning4J



Contents Continued...

- FeedForward Neural Networks Explained
- Abalone Lab
- Review of the Training Web UI
- Convolutional Neural Networks
- Convolutional Network Lab
- Modeling Sequences
- Data Ingest Case Study: Text
- Introduction to Recurrent Neural Networks
- LSTM Character Generation of Weather Forecast Lab
- Saving and Loading trained Models
- Lab: Saving and Loading trained Models



Introductions

- Intro to Skymind
- Instructor Intro
- Student Intro



- Developer of DeepLearning4J
- Services
 - Support
 - Training



Introductions

- Why are you here?
- What do you want to learn?
- What sections interest you?
- What is your background?



What is Deep Learning

This section describes the fields of Artificial Intelligence, Machine Learning and Deep Learning and how they are related.

What is Deep Learning?

The relationship between Deep Learning and Machine Learning

- Machine Learning
- Data Science / Data Mining
- Deep Learning



Introduction to Machine Learning

- ⇒ Machine Learning
- Data Science / Data Mining
- Deep Learning



Machine Learning

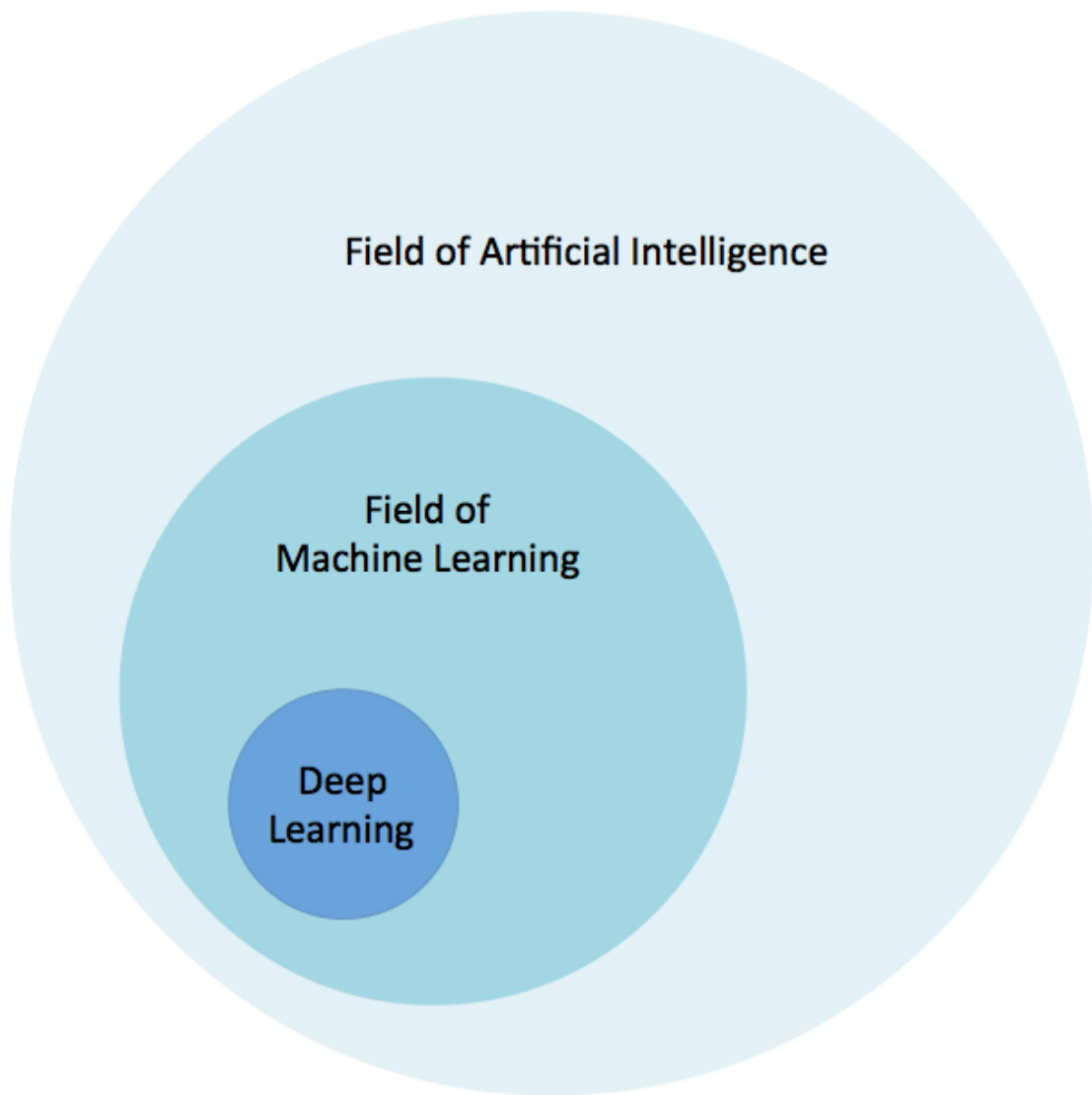


What is Machine Learning?

- 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



See Notes if I have time add pictures

Data Science / Data Mining

- Machine Learning
- \Rightarrow Data Science / Data Mining
- 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



Deep Learning

- Machine Learning
- Data Science / Data Mining
- ⇒ Deep Learning



A Definition of Deep Learning

- Deep learning (also known as deep structured learning, hierarchical learning or deep machine learning) is a branch of machine learning based on a set of algorithms that attempt to model high level abstractions in data.

source - wikipedia



Neural Networks

- A computational approach patterned on the human brain and nervous system



Comparison Between Neural Network and Machine Learning

- Machine Learning
 - Hand Crafted Features
 - SME is needed
 - Must inject Context
- Deep Learning/Neural Network
 - Automatic Feature Engineering
 - Learns Context



Biological Neurons

- Biological Neuron: An electrically excitable cell that processes and transmits information through electrical and chemical signals
- Biological Neural Network: An interconnected group of neurons



Role of Artificial Neural Network

Learns or Trains to perform tasks that traditional programming methods find rather challenging.

- Speech recognition
- object recognition
- computer vision
- pattern recognition.



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
- Neural Networks can be used for either supervised or unsupervised learning

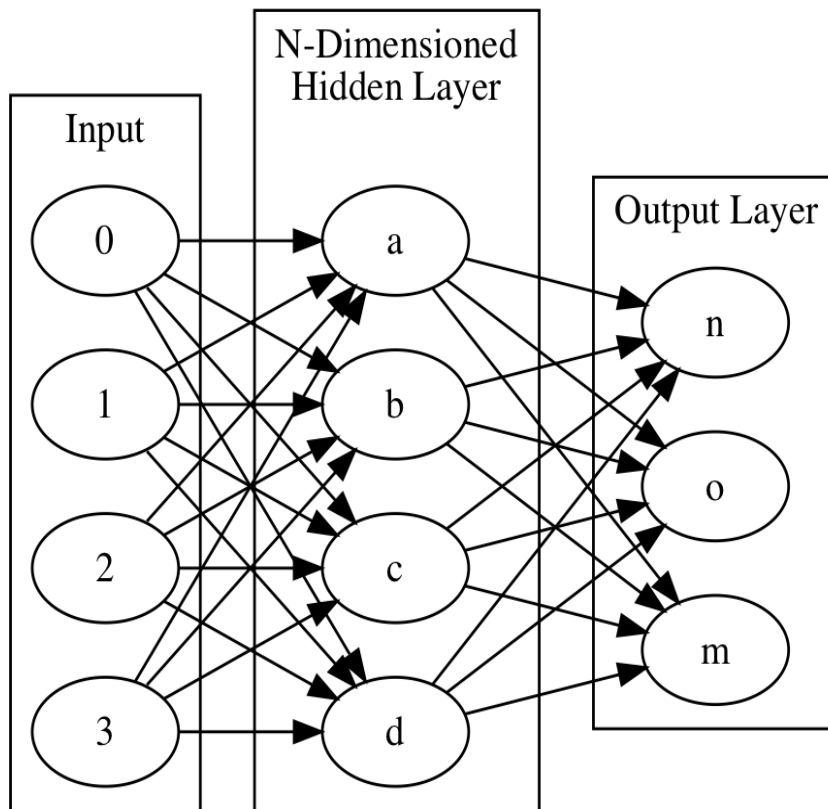


Clustering

- Typically unsupervised learning
 - “K-Means Clustering”
- Example
 - “cluster K groups of similar news articles together”
- ND4J supports this, but it is not a NN



A Neural Network



Framing the Question

Using Neural Networks

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



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

	(Training Records) A				(Parameter Vector) x		(Actual Outcome) b
Input Record 1	0.7500000000000001	0.4166666666666663	0.702127659574668	0.5652173913043479	?		1.0
Input Record 2	0.6666666666666666	0.5	0.9148936170211785	0.6956511739130436	?		2.0
Input Record 3	0.45833333333333326	0.3333333333333336	0.8085106382978723	0.7391304347826088	?		2.0

• =



Linear Algebra Terms

- 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
- Matrices
 - 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)



Neural Networks Use an Iterative Method to Solve a System of Equations

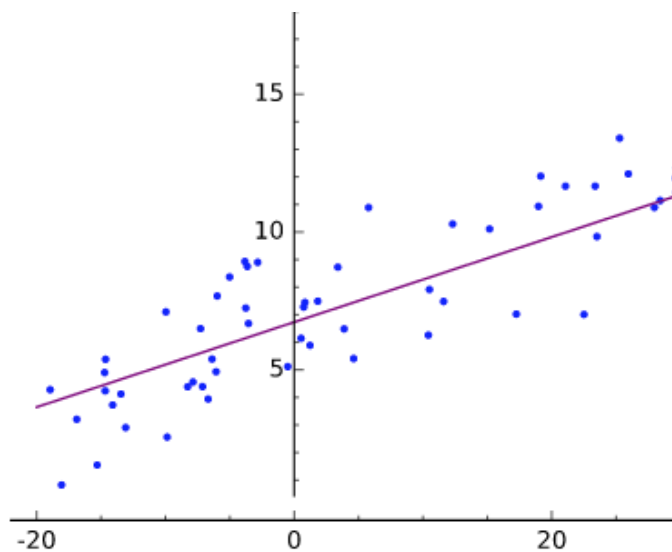


Training a Neural Net

- Inputs: Data you want to produce information from
- Connection weights and biases govern the activity of the network
- Learning algorithm changes weights and biases with each learning pass



Fitting the Training Data

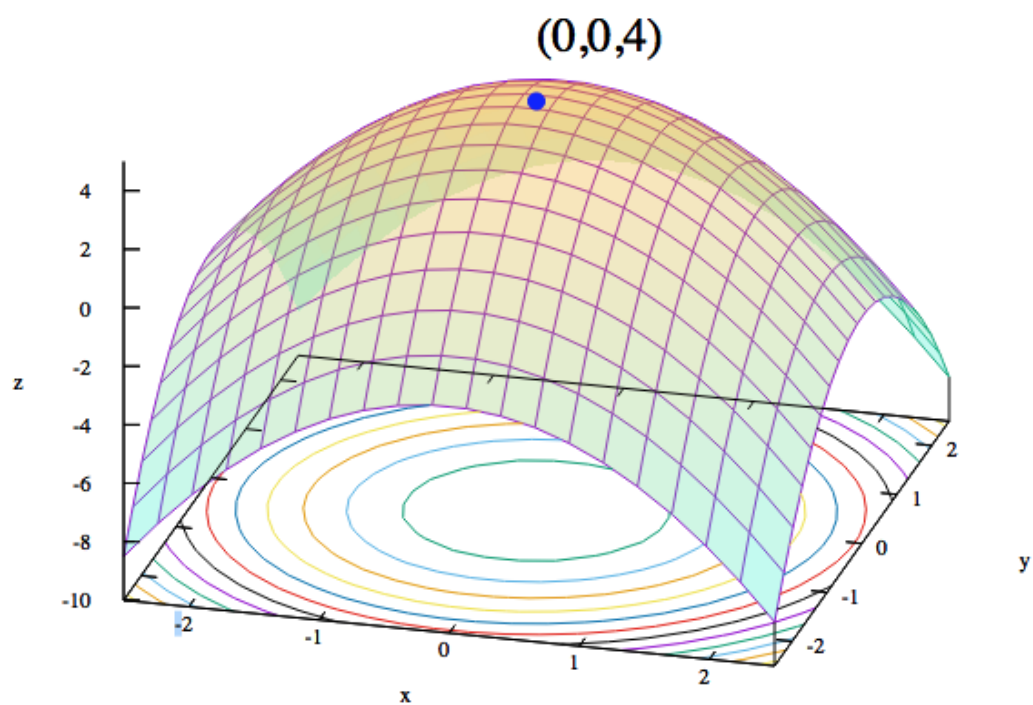


Optimization

- 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

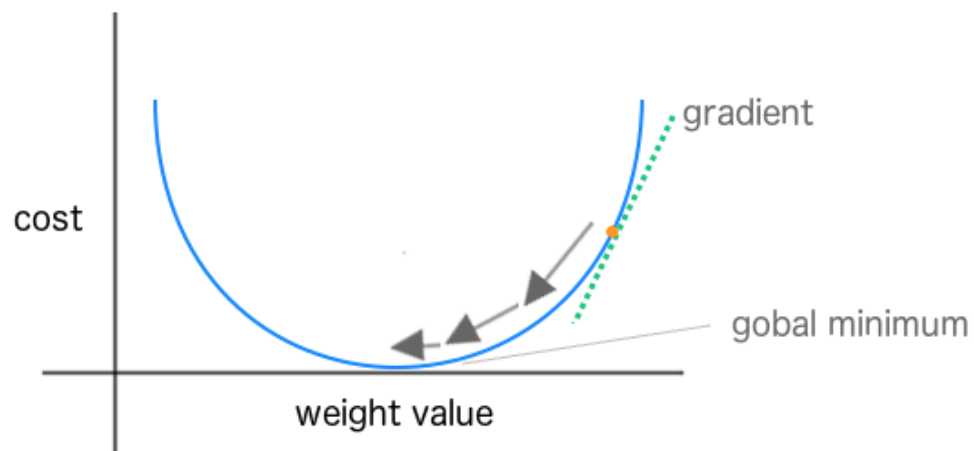


Stochastic Gradient Descent

- 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



SGD Visually Explained



Simple Neural Network Introduction

This Section introduces Neural Networks starting with a very basic example

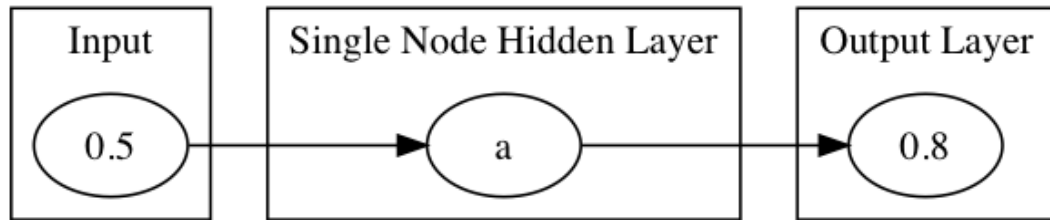
Simple Neural Network

- Simple Network Details
- Simple Network Lab
- Important Network Settings
- VGG-16 Demo



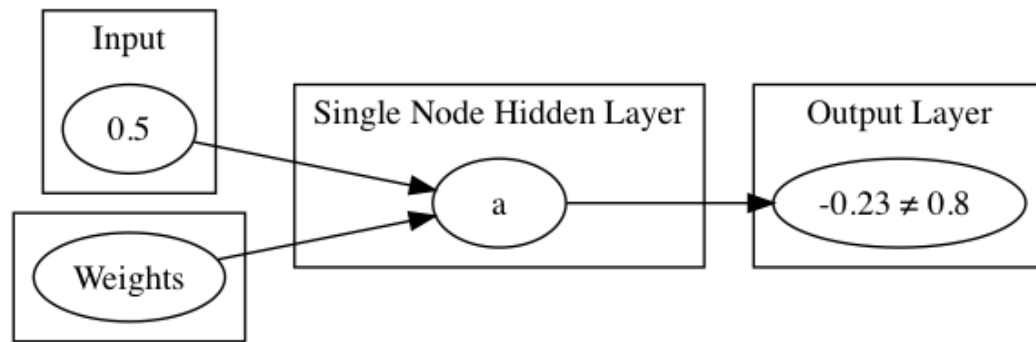
Simple Network

The Goal: Input=0.5 Output=0.8



The Design:

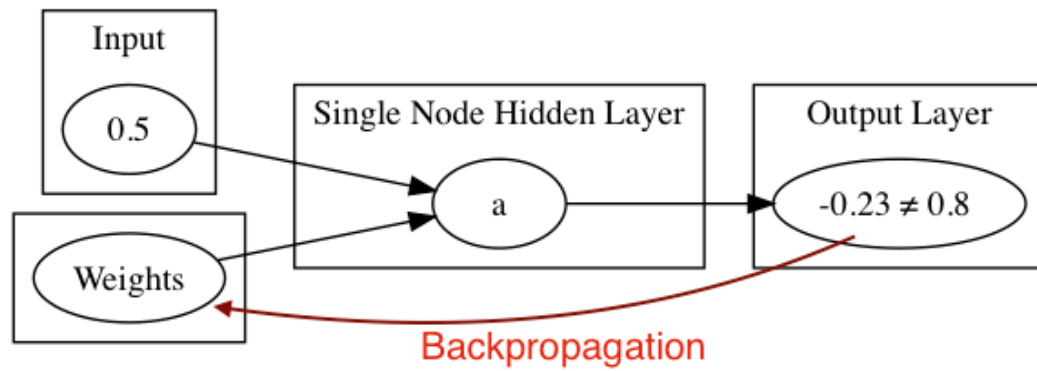
- Apply Random weights, plus activation function per neuron => output



- Test output: does $-0.23 = 0.8$?

Calculate error

- backpropagate to adjust weights
- repeat



Output Layer of a Neural Network Can be Configured for Different Purposes:

- MultiNomial
- Binary
- Continuous
- More...



Simple Network Settings

Simple Network has Settings that will become familiar as they are used in all Neural Networks

- LearningRate
- Updater
- Output Activation
- Hidden Layer Activation
- Number of Epochs
- Loss Function



Extrapolate from Simple Network

- Simple Network took one value in and one value out and trained the network to learn the correct value
- More complex data same process
- Input can be images, row data, documents
- Output can be T/F, range of values, labels of a class



Simple Network Lab

- See your Lab Manual



VGG-16 Demo

- Instructor demonstration of a more complex Neural Network



Questions

Why do we do deep learning?

Which requires Expert Feature Engineering?

A. Deep Learning B. Machine Learning



DeepLearning4J Training UI

Visualizing Network Training with the Deeplearning4j Training UI

- Browser Based
- Real Time



Using the Training UI

- Add Dependency to Pom.xml
- Add code to your class



Maven and pom.xml

- Maven is a dependency management and build tool
- Maven Configuration stored in pom.xml
- Maven Repository stored locally in ~/.m2



pom.xml

```
<dependency>
  <groupId>org.deeplearning4j</groupId>
  <artifactId>deeplearning4j-ui_2.10</artifactId>
  <version>${dl4j.version}</version>
</dependency>
```



Code for UI

```
//Initialize the user interface backend
UIServer uiServer = UIServer.getInstance();

//Configure where the network information
//(gradients, score vs. time etc) is to be stored.
//Here: store in memory.
StatsStorage statsStorage = new InMemoryStatsStorage();
//Alternative: new FileStatsStorage(File), for saving and loading later

//Attach the StatsStorage instance to the UI:
//this allows the contents of the StatsStorage to be visualized
uiServer.attach(statsStorage);

//Then add the StatsListener
//to collect this information from the network, as it trains
net.setListeners(new StatsListener(statsStorage));
```



Viewing the UI

- <http://localhost:9000/train>



Configuring the UI

- System Property org.deeplearning4j.ui.port
- Switch Port

`-Dorg.deeplearning4j.ui.port=9001`



UI execution

- Information is collected and routed to the UI when you call the fit method on your network.



Code Example

<https://github.com/deeplearning4j/dl4j-examples/blob/master/dl4j-examples/src/main/java/org/deeplearning4j/examples/userInterface/UIExample.java>





Using the Overview Page

- Top left: score vs iteration chart - this is the value of the loss function on the current minibatch
- Top right: model and training information
- Bottom left: Ratio of parameters to updates (by layer) for all network weights vs. iteration
- Bottom right: Standard deviations (vs. time) of: activations, gradients and updates





Model Page

- On the right, the following charts are available, after selecting a layer:
 - Table of layer information
 - Update to parameter ratio for this layer, as per the overview page. The components of this ratio (the parameter and update mean magnitudes) are also available via tabs.
 - Layer activations (mean and mean \pm 2 standard deviations) over time
 - Histograms of parameters and updates, for each parameter type
 - Learning rate vs. time (note this will be flat, unless learning rate schedules are used)



Deeplearning4J UI and Spark Training

- Conflict with Spark Dependencies
- Solution
 - Collect and save the relevant stats, to be visualized (offline) at a later point
 - Run the UI in a separate server, and Use the remote UI functionality to upload the data from the Spark master to your UI instance



Offline Use

- Collecting Stats

```
SparkDL4jMultiLayer sparkNet = new SparkDL4jMultiLayer(sc, conf, tm);
```

```
StatsStorage ss = new FileStatsStorage(new File("myNetworkTrainingStats.dl4j"));  
sparkNet.setListeners(ss, Collections.singletonList(new StatsListener(null)));
```

- Load for Later use

```
StatsStorage statsStorage = new FileStatsStorage(statsFile);  
//If file already exists: load the data from it UIServer uiServer = UIServer.getInstance();  
uiServer.attach(statsStorage);
```



Remote UI

- In JVM running the UI:

```
UIServer uiServer = UIServer.getInstance(); uiServer.enableRemoteListener();  
//Necessary: remote support is not enabled by default
```

- In the Spark Training instance:

```
SparkDI4jMultiLayer sparkNet = new SparkDI4jMultiLayer(sc, conf, tm); StatsStorageRouter  
remoteUIRouter = new RemoteUIStatsStorageRouter("http://UI_MACHINE_IP:9000");  
sparkNet.setListeners(remoteUIRouter, Collections.singletonList(new StatsListener(null)));
```



Using UI to Tune Your Network

- Score vs. Iteration
- Goal: Decrease
- If Increasing
 - Reduce Learning Rate
 - Incorrect Normalization?
- Slow decrease or Flat?
 - Learning Rate to Low
 - Optimization Challenges, try or adjust adaptive updater
 - Nesterovs (momentum), RMSProp or Adagrad
- Rough or Abnormal Graph
 - Verify Data is shuffled



Neural Network Internals

This section describes key configuration details of Neural Networks

Neural Network Key Concepts

- Activation Functions
- BackPropagation
- Loss Functions
- Weight Initialization
- Data Normalization and Standardization



Activation Functions

What is an Activation Function

- Determines output of Neuron Based on Inputs
- Non-Linear Transform function at each node
- Defined per layer
- Allow neural networks to make complex boundary decisions for features at various levels of abstraction.



Activation Function

- Hidden Layer
- Controls what is sent to connected Neurons
- Output Layer
- Determines Output Value
 - Regression
 - Classification
 - Etc












Common Activation Functions

- ReLU
 - Recent Breakthrough
 - Good Default
- Sigmoid
 - Had been Standard
 - S curve Range 0 to 1
- TanH
 - S curve Range -1 to 1



Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU)		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Output Layer Activation

- Special Case
- Goal of hidden Layer activation is to squash intermediate values
- Goal of output Layer is to answer our question
 - Classification = softmax
 - regression = identity



Output Layer Guidelines

- Classification
 - softmax activation
 - Negative Log Likelihood for loss Function
 - Multi Class Cross Entropy
- Softmax
 - Probability Distribution over classes
 - Outputs sum to 1.0
- Regression
 - Identity Activation
 - MSE(Mean Squared Error) Loss Function



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
 - $P(\text{Heads}) = 0.5$ every time



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



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



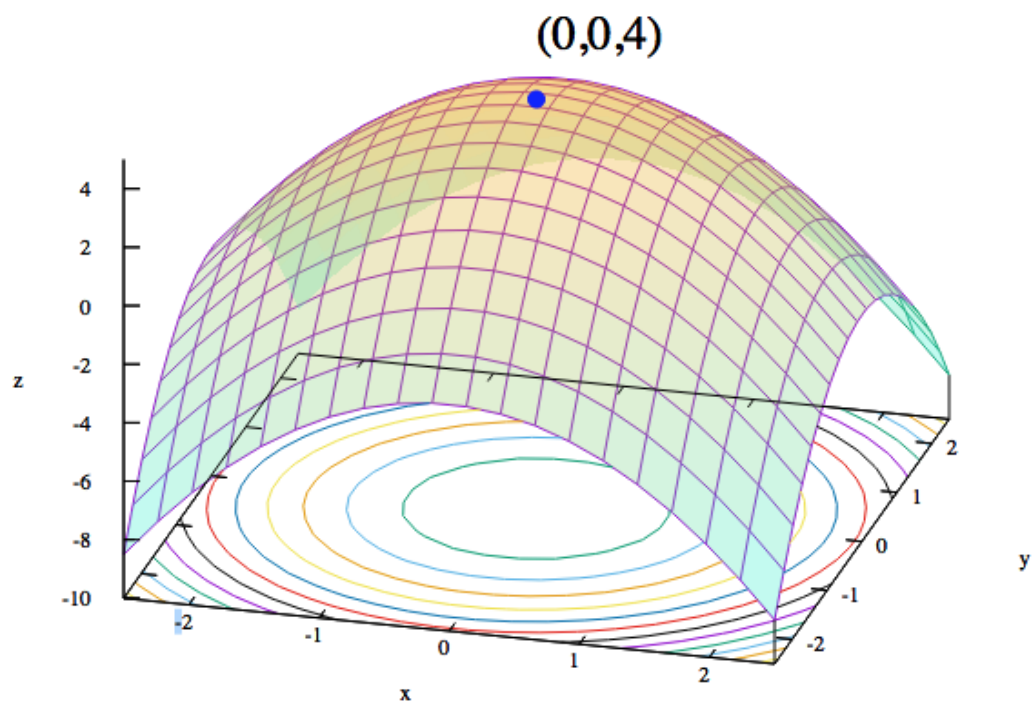
Back Propagation and Updaters/optimizers

What is BackPropagation

- The process of updating the weights of a Neural Network to reduce error
- The Forward Pass generates an output
- The Loss Function calculates the error
- Gradient Descent is used to minimize the error
- Steps are repeated and network continues to improve



Gradient Descent in Weight Space



Adapting to Changing Error

- Initially error will be large
- Output more or less random
- Two approaches
 - Dynamic Learning Rate (Anneal the Learning Rate)
 - Use Adaptive Optimizer



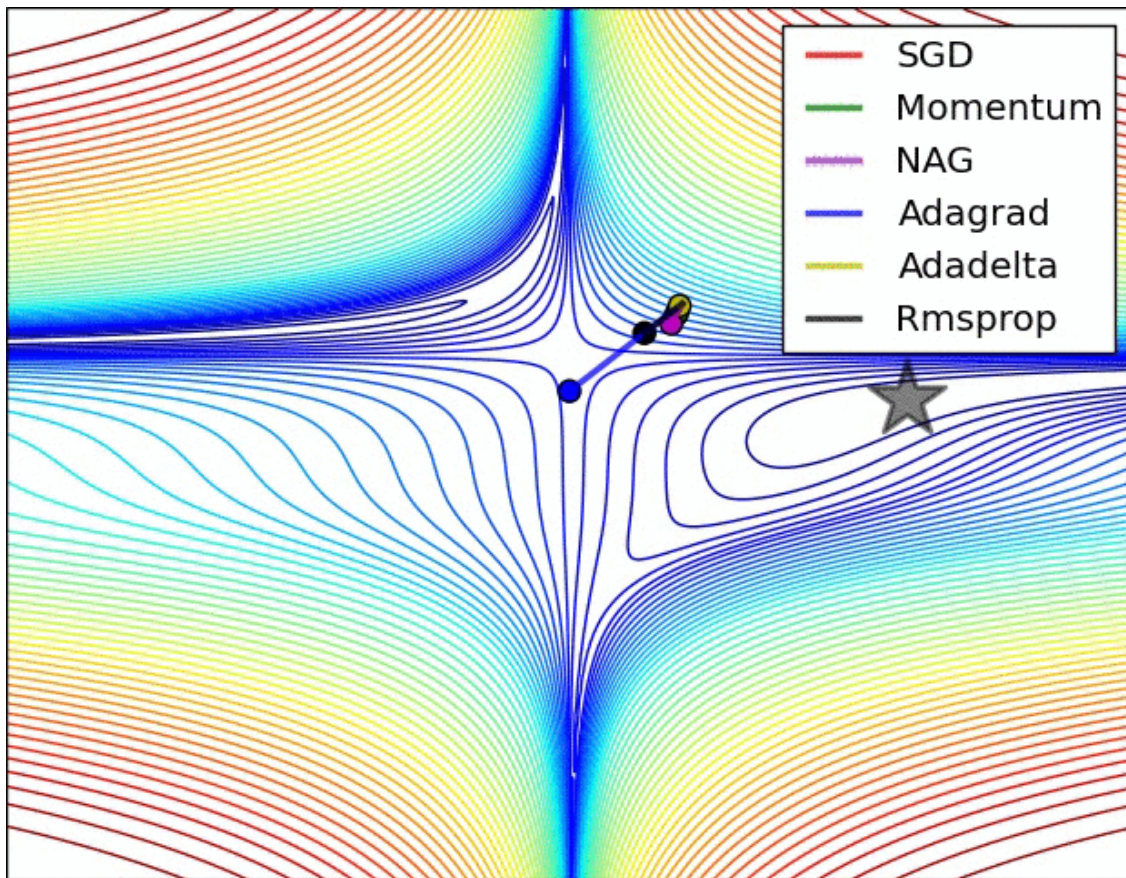
Adaptive Optimizers / Dynamic Learning Rate a Simplified View

- Initially you want the network to train quickly
 - Error is large
 - Take Large Steps
- As Error decreases
 - Smaller steps are better



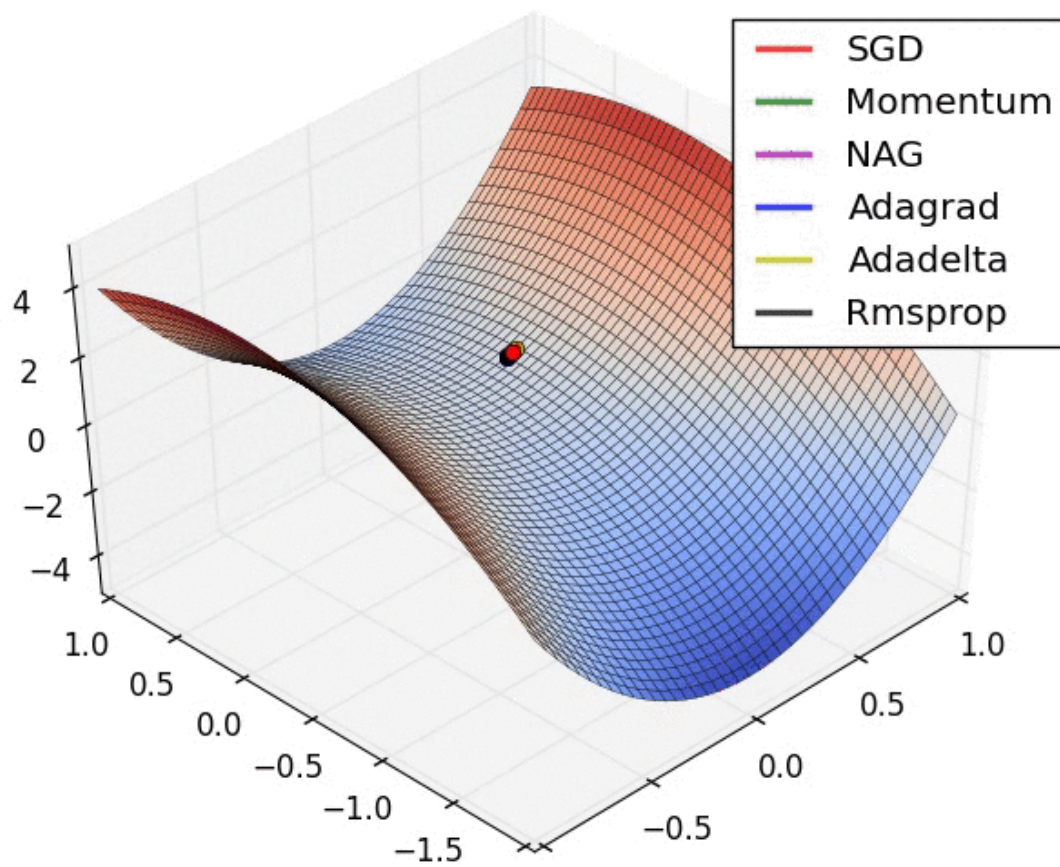
Updater Animation

- Thanks to Alec Radford



Updater Animation

- Thanks to Alec Radford



Loss Functions

Loss functions quantify how close a given neural network is to the ideal it is training towards.

The three important functions at work in machine learning optimization:

- parameters
 - transform input to help determine the classifications a network infers
- loss function
 - gauges correctness of output
- optimization function
 - guides it toward the points of least error.



Available Loss Functions

- For Regression
 - MSE(Mean Squared Error)
 - Mean Absolute Error(MAE)
 - Mean Squared Log Error (MSLE)
 - Mean Absolute Percentage Error(MAPE)



Regression Loss Function Common Usage

- MSLE and MAPE handle large ranges,
 - common practice to normalize input to suitable range and use MSE or MAE



Loss Functions for Classification

- Hinge Loss (SVM)
 - Hard Classification 0,1 a -1,1 classifier
- Logistic Loss (logistic regression)
 - Probabilities per class



Negative Log Likelihood

- Likelihood
 - between 0 and 1
- Log likelihood
 - between negative infinity and 0
- Negative log Likelihood
 - between 0 and infinity



Weight Initialization

Why not initialize all weights to zero?

- Nodes would never diverge
 - All would learn the same thing



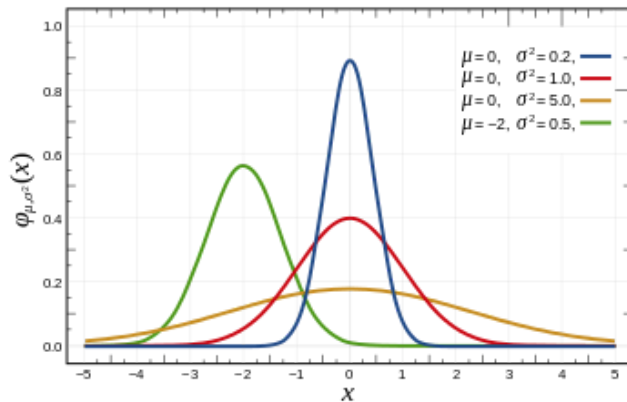
Solution

- Randomize weights
- Mean 0 + some randomized distribution



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



Weight Initialization: The Challenge

- Weights too small
 - Signal shrinks as it passes through layers
 - Becomes too small to be useful
 - Vanishing Gradient Problem
- Weights too large
 - Signal Grows as it passes through Layers
 - Becomes too large to be useful
 - Exploding Gradient Problem



USEful Weight Distribution Techniques

- Truncated Normal
- Xavier
- Relu



Xavier Distribution

- Most Common
- 0 mean and a specific variance
 - $\text{Var}(W) = 1/n \ln$



Benefits of Xavier

- Xavier Enabled Full Network Training vs per-Layer Pre-Training
- Big Breakthrough



Relu

- Relu
- Works well with CNN's and Relu activations



Data Normalization and Standardization

Why Normalize

- Avoid large weights dominating
- In Traditional ML
 - Linear Regression, Logistic Regression
 - Must Normalize
 - Tree Based can cope with large range
- Same reasons as weight initialization



Normalization

- Convert to range of values 0-1
 - Max Value becomes 1
 - Min Value becomes 0



Standardization

- More Common
- Convert Values to
 - mean 0
 - Standard Deviation of 1



Tuning Neural Networks

Hyper-Parameters that may need tuning

- Learning Rate
- Batch Size
- Updater



Learning Rate Guidelines

- 0.1 to 0.000001



Learning Rate

- Adaptive Learning Rate
- Adjust optimizer based on previous updates
 - Nesterovs Momentum
 - Adagrad, Adadelata, Adam, RMSProp



Learning Rate Schedules

- Tune Learning Rate as Learning Progresses
- Based on Schedule or other metrics



Early Stopping

- Stop Training once overfitting is detected



Avoiding Overfitting

- What is Overfitting
 - Scores well on test
 - Scores poorly on unseen examples
- Has "memorized" training data
- Fails to generalize



Regularization

- L1 and L2 Regularization
 - penalizes large network weights
 - avoids weights becoming too large
- Risks
 - Coefficients too high
 - Network stops learning
- Common values for L2 regularization
 - 10^{-3} to 10^{-6} .

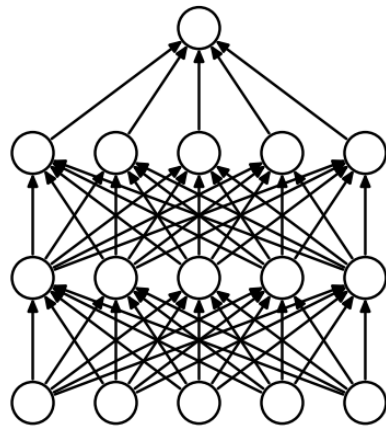


Dropout

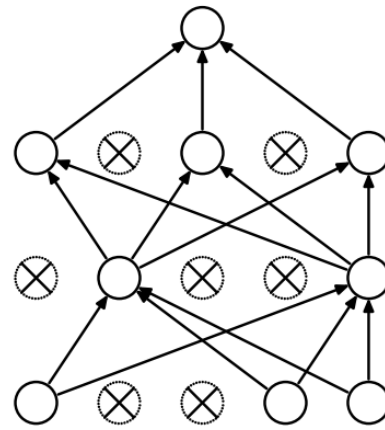
- Set probability that a Neuron will be de-activated, set activation to 0
- Forces Network to learn different redundant representations
- Commonly used dropout rate of 0.5.



Dropout



(a) Standard Neural Net



(b) After applying dropout.

Types of Neural Networks

This section introduces the various types of Neural Networks

- \Rightarrow Inspiration for Neural Networks
- FeedForward Neural Networks
- Recurrent Neural Networks
- Convolutional Neural Networks
- Other



Inspired by Biological Neurons

- Interconnected computational units with output based on input combined with an activation function



The Main Types of Neural Networks

- FeedForward Neural Networks (Multi Layer Perceptron)
- Recurrent Neural Networks
- Convolutional Neural Networks



- Inspiration for Neural Networks
- \Rightarrow FeedForward Neural Networks
- Recurrent Neural Networks
- Convolutional Neural Networks
- Other



FeedForward Neural Network Uses

- Supervised Learning
- Classification
- Regression



FeedForward Neural Network Data

- Tabular data
- Image data



FeedForward Neural Network Output

- Flexible, range of values, one of a class probabilities
- Limitation, one input maps to one output



- Inspiration for Neural Networks
- FeedForward Neural Networks
- \Rightarrow Recurrent Neural Networks
- Convolutional Neural Networks
- Other



Recurrent Neural Networks

- In this class when we talk about RNNs we are typically talking about LSTMs



Recurrent Neural Networks Uses

- Sequence or Time Series Data



Recurrent Neural Networks Input and Output

- One to Many
 - Image to words for caption
- Many to Many
 - French word sequence to English word sequence
- Many to one
 - Speaker identification given voice sample



- Inspiration for Neural Networks
- FeedForward Neural Networks
- Recurrent Neural Networks
- \Rightarrow Convolutional Neural Networks**
- Other



Convolutional Neural Network

- Inspired by the visual cortex
- Useful for image recognition



How Convolutional Neural Networks differ from other Networks

- Image data analyzed as 4D NDArrays (tensors)
- For a 10*10 image, a series of 3*3, or 5*5 subsets are analyzed



- Inspiration for Neural Networks
- FeedForward Neural Networks
- Recurrent Neural Networks
- Convolutional Neural Networks
- \Rightarrow Other



Other Useful Neural Network Types

Not covered in this course

- Variational Autoencoders
- Restricted Boltzman Machines
- more....



Variational Autoencoders

- Unsupervised learning of features
- Input -> Neural Net -> rebuild input
- Reduction in dataset dimensionality



Restricted Boltzmann Machines

- Feature extraction and dimensionality reduction
- Model probability
- Useful for the pre-training phase in other large Deep Networks



DeepLearning4J

- DL4J overview
- DataVec
- ND4J and LibND4J
- DeepLearning4j

DeepLearning4J Overview

Goals of the DeepLearning4J project

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

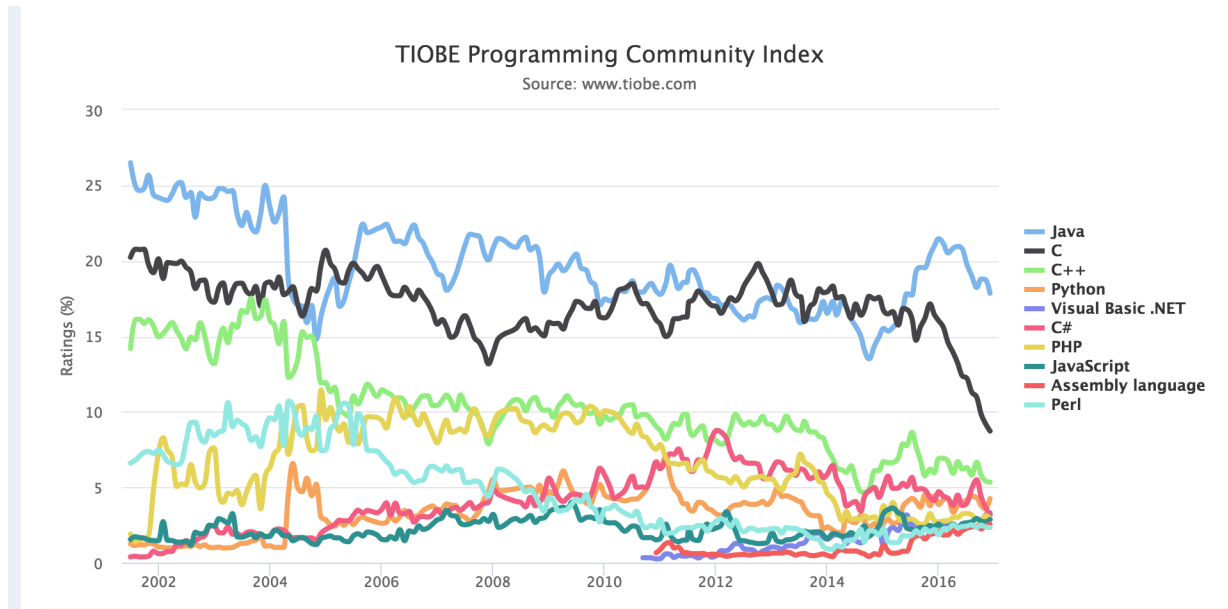


DeepLearning4J sub-projects

- DataVec
 - Tools for ETL
- ND4J
 - NUmeric Arrays
 - NumPY for the JVM
- libnd4j
 - Native Libraries for GPUs/CPUs
- DeepLearning4J
 - Tools to train Neural Networks



Why Java?



DataVec

- Neural networks process numeric arrays
- Datavec helps you get from `your_data` => numeric array

Data Sources

- Log files
- Text documents
- Tabular data
- Images and video
- and more !!



Goal

- Build a user freindly comprehensive toolkit for data pilelines into Neural Netowrks



DataVec Features

- Transformation
- Scaling
- Shuffling
- Joining
- Splitting Test and Train

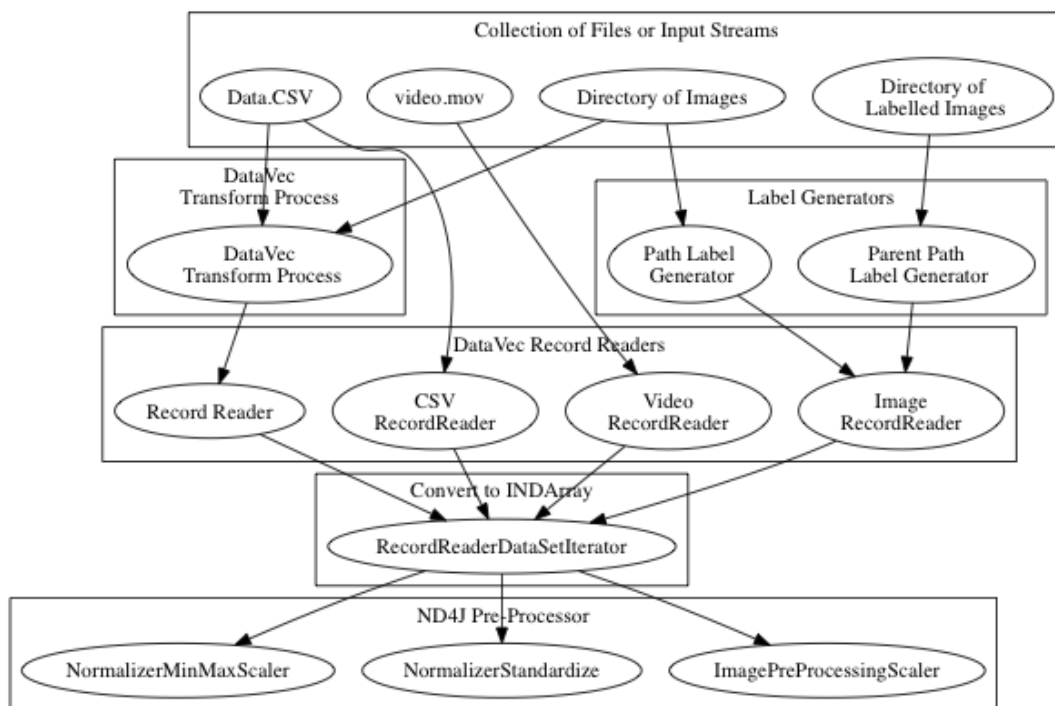


Commonly Used Features

- RecordReaders
 - Read files or input, convert to List of Writables
- Normalizers
 - Standardize, scale or normalize the data
- Transform Process
 - Join datasets, replace strings with numerics, extract labels

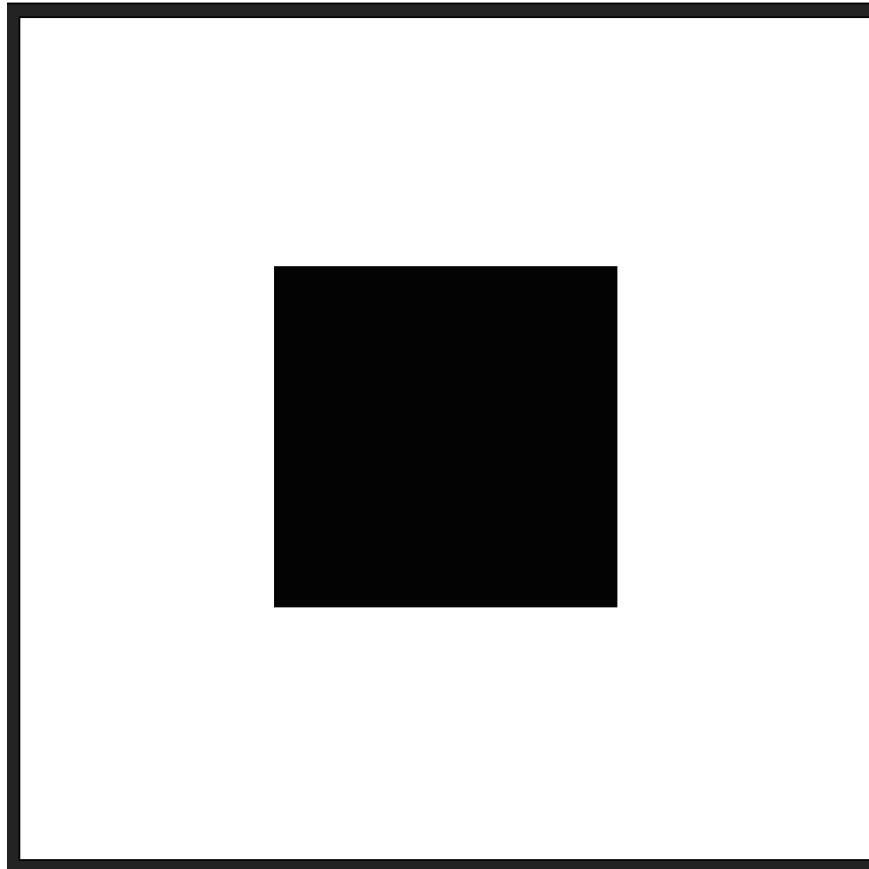


Diagram of available ETL paths



DataVec Image Basics

- Images are arrays of pixel values



Code Example: Load above image as INDArray

```
INDArray imagematrix = loader.asMatrix(image); System.out.println(imagematrix);
```

Output

```
[[[255.00, 255.00, 255.00, 255.00], [255.00, 0.00, 0.00, 255.00], [255.00, 0.00, 0.00, 255.00], [255.00, 255.00, 255.00, 255.00]]]
```



Code Example: Scale values between 0 and 1

```
DataNormalization scaler = new ImagePreProcessingScaler(0,1);  
scaler.transform(imagematrix);
```

Output

```
[[[[1.00, 1.00, 1.00, 1.00], [1.00, 0.00, 0.00, 1.00], [1.00, 0.00, 0.00, 1.00], [1.00,  
1.00, 1.00, 1.00]]]]
```



Manipulating Images with DataVec

- Scale images to same dimensions with RecordReader
- Used in pipeline for example in training

```
ImageRecordReader recordReader = new ImageRecordReader(height,width,channels);
```

- Scale image to appropriate dimensions with NativeImageLoader
- Used one-off for example in inference

```
NativeImageLoader loader = new NativeImageLoader(height, width, channels); \ load and  
scale INDArray image = loader.asMatrix(file); \ create INDArray INDArray output =  
model.output(image); \ get model prediction for image
```



Code Example: Image Transform

- Scale pixel values

```
DataNormalization scaler = new ImagePreProcessingScaler(0,1); scaler.fit(dataIter);  
dataIter.setPreProcessor(scaler);
```



Image Data Set Augmentation

- Create "larger" training set with OpenCV/dataVec tools
 - Transform
 - Crop
 - Skew



Applying Labels

- ParentPathLabelGenerator
- PathLabelGenerator



Available Record Readers

- Table of available record readers:
 - <https://deeplearning4j.org/etl-userguide>



Available ND4J Pre-Processors

- ImagePreProcessingScaler
 - min max scaling default 0 + - 1
- NormalizerMinMaxScaler
 - Scale values observed min -> 0, observed max -> 1
- NormalizerStandardize
 - moving column wise variance and mean
 - no need to pre-process



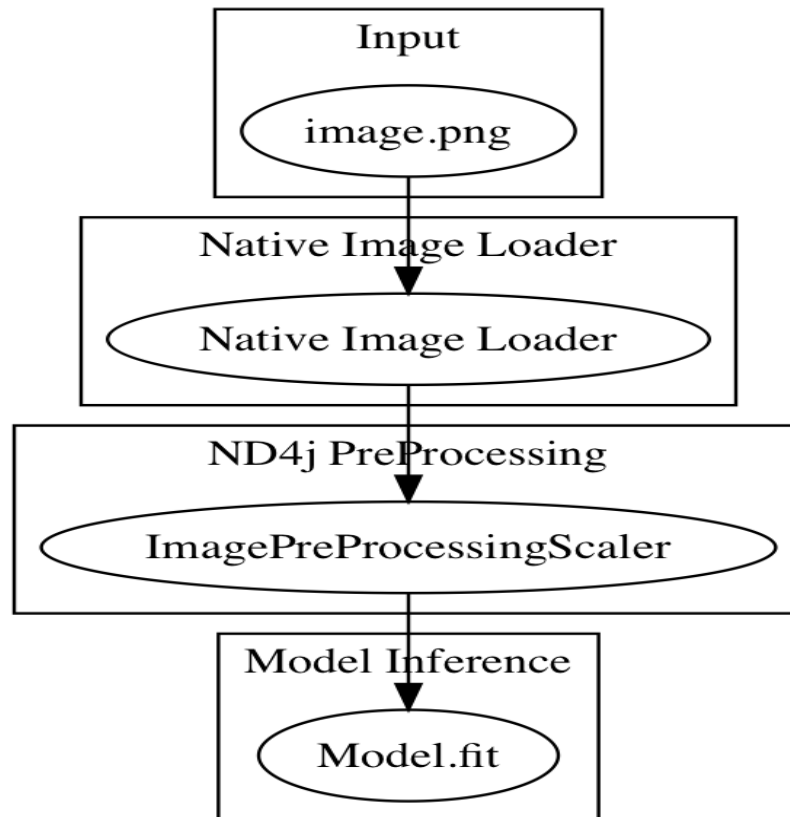
Image Transforms with JavaCV, OpenCV, ffmpeg

- Libraries included



Make this first intro to this for inference, and clean order

Image pipeline Single Image to Pre-Trained Model



Code Example: CSV Data to INDArray

```
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 handles loading/parsing
        int numLinesToSkip = 0;
        String delimiter = ",";
        RecordReader recordReader =
            new CSVRecordReader(numLinesToSkip, delimiter);
        recordReader.initialize(new FileSplit
            (new ClassPathResource("iris.txt").getFile()));
    }
}
```



Code Example: Continued....

```
//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 input features followed by an integer label (class) index.
//Labels are the 5th value (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.
//Loading 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 `Writable`s
 - 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`
 - `INDArray` of Features, `INDArray` of Labels



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



DataVec Lab

See the Lab Manual for the DataVec Lab

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

- Classes frequently Used
 - DataSet
 - Container for INDArrays of Features/Labels
 - DataSetIterator
 - Build DataSet from RecordReader



libND4J

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



FeedForward Neural Networks Explained

FeedForward Neural Networks

- Share many features with Convolutional and Recurrent Neural Networks
- FeedForward neural networks ~ MultiLayerPerceptrons
- Developed in the 1940s-1960s

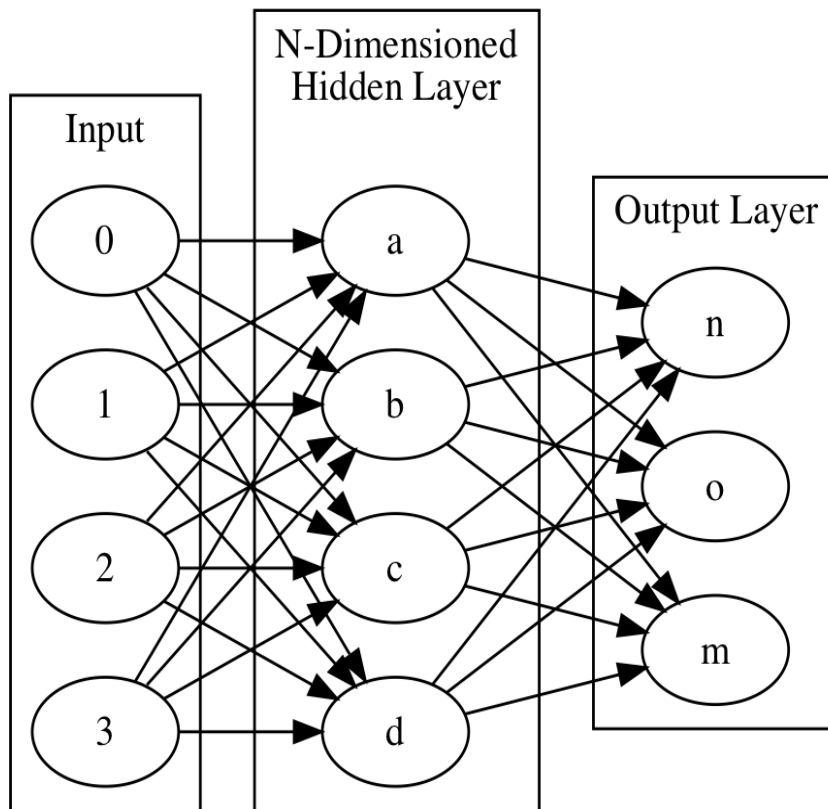


Why Deep Learning now

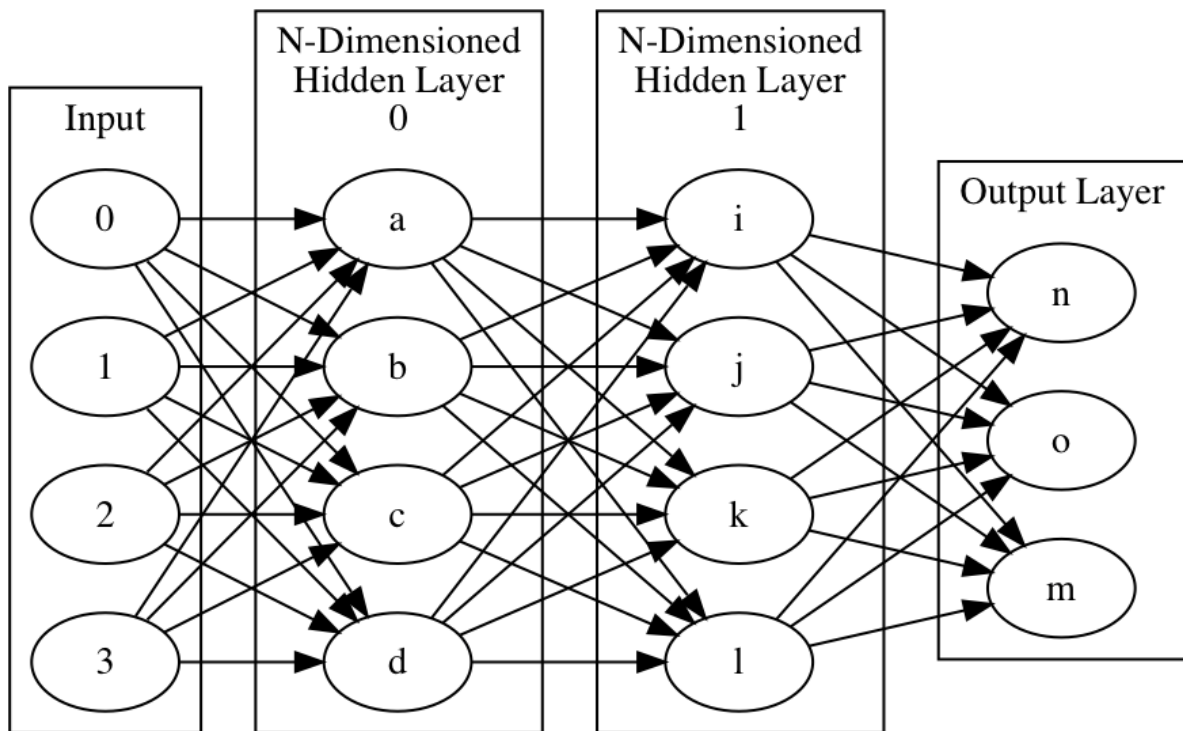
- Successful use came with the rise of GPUs and larger training sets



A FeedForward Neural Network



A Neural Network with Two Hidden Layers



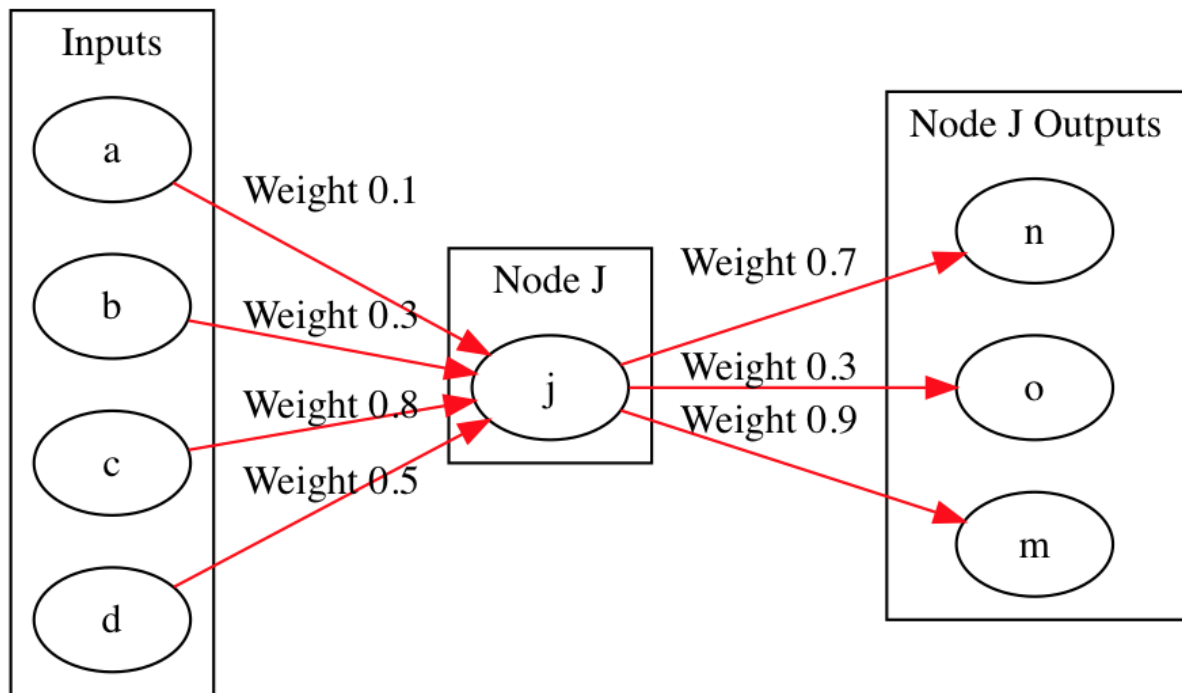
Connections Between Nodes

- Designed to be selective and trainable
- Filter, aggregate, convert, amplify, ignore what they pass on to next neuron
- This transformation converts raw input into useful information
- Input has trainable weight applied
- Output determined by activation function



Neural Network Diagram Explained

- Node J



Single Node Diagram Discussion

- Input
 - Input is determined by the output of the input neurons * the weights applied to that output
- Weights
 - Assigned randomly* initially between 0-1
 - Weight of 0, input is ignored
 - Large weight input is amplified
 - As the network trains weights are adjusted
- Output
 - Output is determined by its input * weights, and the activation function.
 - Sigmoid activation low input output 0 higher input output 1, in between S curve.
 - ReLU low input 0 then linear after trigger.



Key Terms

- Activation Function
 - A nonlinear A function that maps input on a nonlinear scale such as sigmoid or tanh. By definition, a nonlinear function's output is not directly proportional to its input
- Loss Function
 - How error is calculated
- Weights
- BackProp



Neural Net Terms

- Weights Connection Weights. Weights on connections in a neural network are coefficients that scale (amplify or minimize) the input signal to a given neuron in the network
- Activation Function
 - The sigmoid function. "Its best thought of as a *squashing function*, because it takes the input and squashes it to be between zero and one: Very negative values are squashed towards zero and positive values get squashed towards one." Karpathy



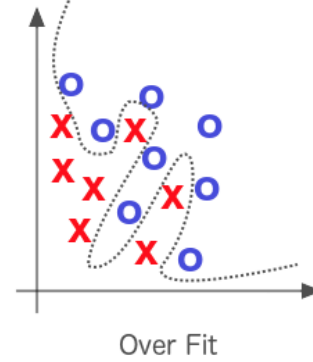
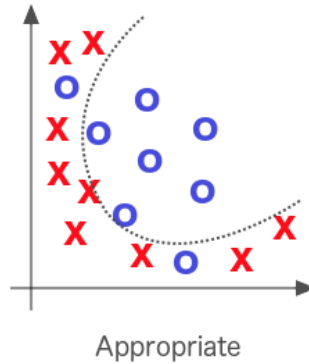
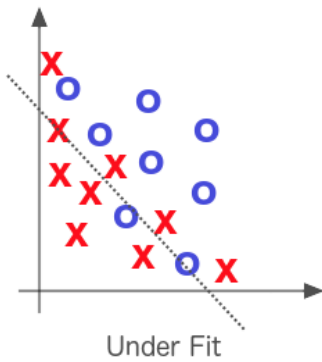
Training a Neural Net

- Inputs: Data you want to produce information from
- Connection weights and biases govern the activity of the network
- Learning algorithm changes weights and biases with each learning pass



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



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



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”



Confusion Matrix

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

LAB: FeedForward Network to Determine Age of Shellfish based on Measurements

Please refer to your lab manual.

Data Ingestion Case Study: Text

Table of Contents

- PreProcessing and tokenization
- Bag of Words
- N-Grams
- Word2Vec
- Paragraph Vectors
- GloVE
- Words as Sequence of Characters



Table of Contents

- ⇒ PreProcessing and tokenization
- Bag of Words
- N-Grams
- Word2Vec
- Paragraph Vectors
- GloVE
- Words as Sequence of Characters



PreProcessing and Tokenization

- Tokenizer
 - Splits stream of words into individual words
 - DefaultTokenizer
 - NGramTokenizer
 - PosUimaTokenizer
 - UimaTokenizer
- PreProcessors
 - LowCasePreProcessor
 - StemmingPreprocessor



Table of Contents

- PreProcessing and tokenization
- \Rightarrow Bag of Words
- N-Grams
- Word2Vec
- Paragraph Vectors
- GloVE
- Words as Sequence of Characters



Bag of Words

Corpus is represented as the bag(multiset) of its words.

- No Grammar
- No order
- Frequency only

"Bob and Carol and Ted and Alice"

Becomes the List ["Bob","and","Carol","Ted","Alice"]

Term frequency [1,3,1,1,1]



Bag of Words Uses

- TfIDF
 - Frequency of word/document compared to word/corpus of documents



Bag of Words Example

- Lab folder has example
- Tokenizer to read files from directory and label with filename

```
TokenizerFactory tokenizerFactory = new DefaultTokenizerFactory();
LabelAwareIterator iterator = new FilenamesLabelAwareIterator
    .Builder()
        .addSourceFolder(new ClassPathResource("bow").getFile())
        .useAbsolutePathAsLabel(false)
        .build();
```



Bag of Words Example

- Code to show contents of iterator

```
while(iterator.hasNext()){  
    LabelledDocument doc = iterator.nextDocument();  
    System.out.println(doc.getContent());  
    System.out.println(doc.getLabels().get(0));  
}  
  
iterator.reset();
```



Bag of Words Example

```
BagOfWordsVectorizer vectorizer = new BagOfWordsVectorizer.Builder()  
    .setMinWordFrequency(1)  
    .setStopWords(new ArrayList<String>())  
    .setTokenizerFactory(tokenizerFactory)  
    .setIterator(iterator)  
    .build();  
vectorizer.fit();
```



Bag of Words Example

- Code to explore the contents of the Bag of Words

```
log.info(vectorizer.getVocabCache().tokens().toString());  
System.out.println(vectorizer.getVocabCache().totalNumberOfDocs());  
System.out.println(vectorizer.getVocabCache().docAppearedIn("two."));  
System.out.println(vectorizer.getVocabCache().docAppearedIn("one."));  
System.out.println(vectorizer.getVocabCache().docAppearedIn("world"));
```



Table of Contents

- PreProcessing and tokenization
- Bag of Words
- \Rightarrow N-Grams
- Word2Vec
- Paragraph Vectors
- GloVE
- Words as Sequence of Characters



- Contiguous sequence of n items from a sequence of text
- Example text ***"It is the year 2016"***
 - Extracted Bi-grams
 - "It is"
 - "is the"
 - "the year"
 - "year 2016"
 - Extracted Tri-grams
 - "It is the"
 - "is the year"
 - "the year 2016"

NGram uses

- Provide more context than Bag of Words
- Used in some neural networks for speech recognition to narrow the scope of prediction
 - RNN predicts next word out of top x percent of trigram for previous 2 word predictions



Code Example: NGrams in DL4J

```
public static void main(String[] args) throws Exception{
    String toTokenize = "To boldly go where no one has gone before.";
    TokenizerFactory factory =
        new NGramTokenizerFactory(new DefaultTokenizerFactory(), 1, 2);

    Tokenizer tokenizer = factory.create(toTokenize);
    factory = new NGramTokenizerFactory
        (new DefaultTokenizerFactory(), 2, 3);

    List<String> tokens = factory.create(toTokenize).getTokens();
    log.info(tokens.toString());
}
```

Output

```
[To, boldly], [boldly, go], [go, where],.....
[To, boldly, go], [boldly, go, where] .....
```



Table of Contents

- PreProcessing and tokenization
- Bag of Words
- N-Grams
- \Rightarrow Word2Vec
- Paragraph Vectors
- GloVE
- Words as Sequence of Characters



Word2Vec

- Model for word embeddings
- Vector Space
- Each word in corpus \Rightarrow vector in multi-dimensional vector space
- Relative location of word in vector space denotes relationship
 - Distance and direction from Boy- \rightarrow Man
 - Distance and Direction from Girl- \rightarrow Woman



Word2Vec

- Model for word embeddings
- Vector Space
- Each word in Corpus \Rightarrow Vector in Vector Space



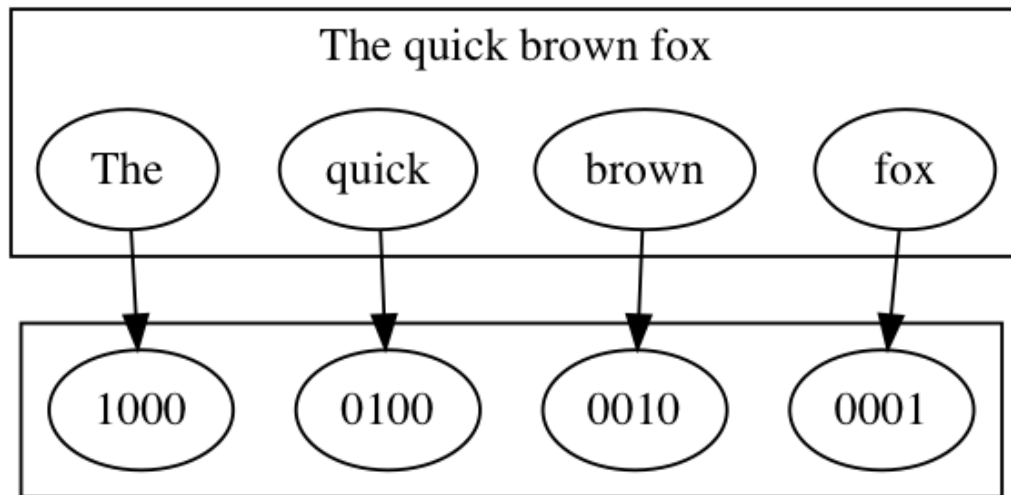
Word2Vec - Generating the Vector Space

- Neural Network trained to return word probabilities of a moving window
 - Given word "Paris", out of the corpus of words predict probability of each word occurring within say five words of the word "Paris"
- One hot Vector, size of every word in the corpus
- all 0's except for 1 representing the word
- See Demo <https://ronxin.github.io/wevi/>
- See example in intellij
- Allows you to do word math
 - $\text{King} - \text{Man} + \text{Woman} = (?) \text{Queen}$



One-hot encoding

- Vector, the size of the vocabulary, all 0 except for single 1

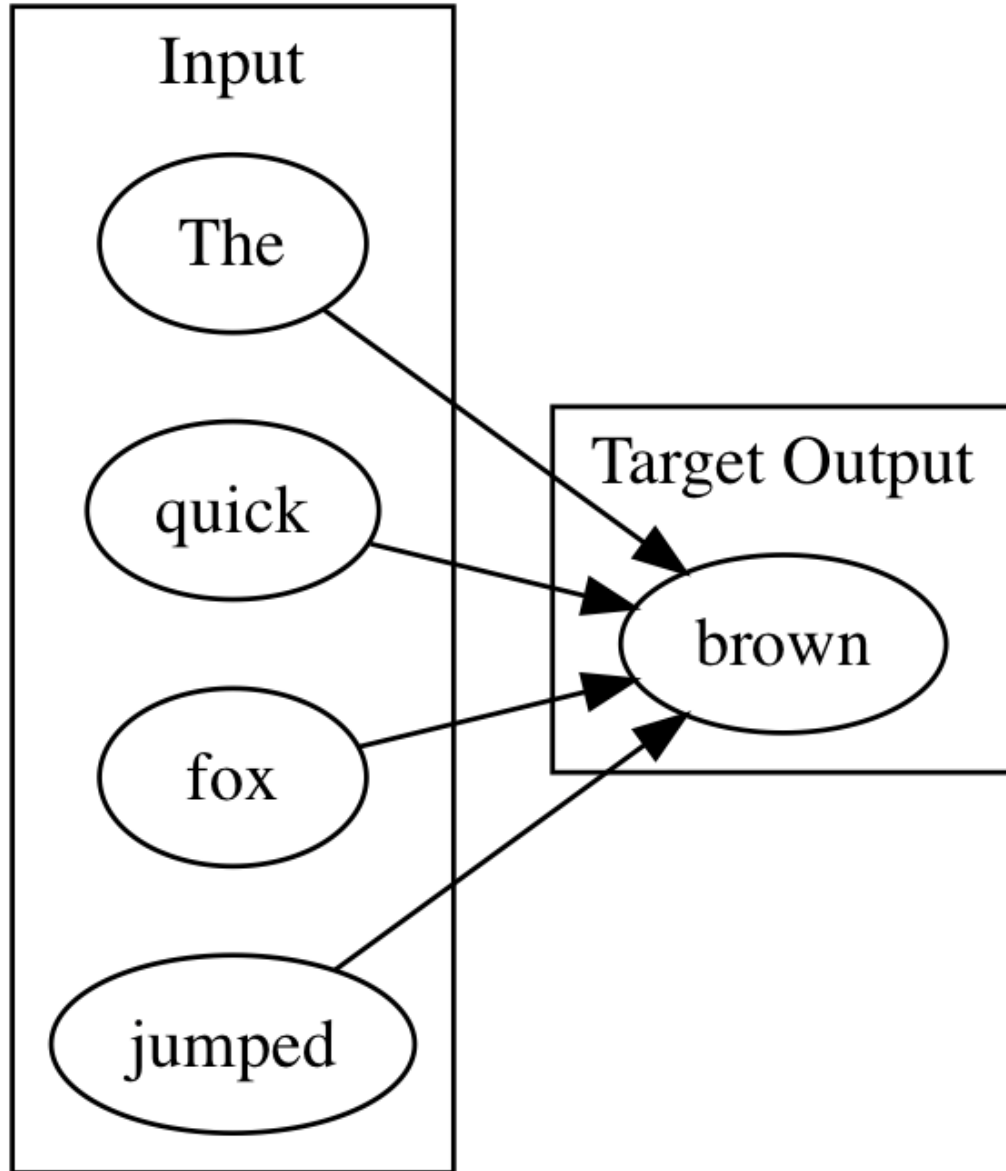


Two Methods for Building word2vec

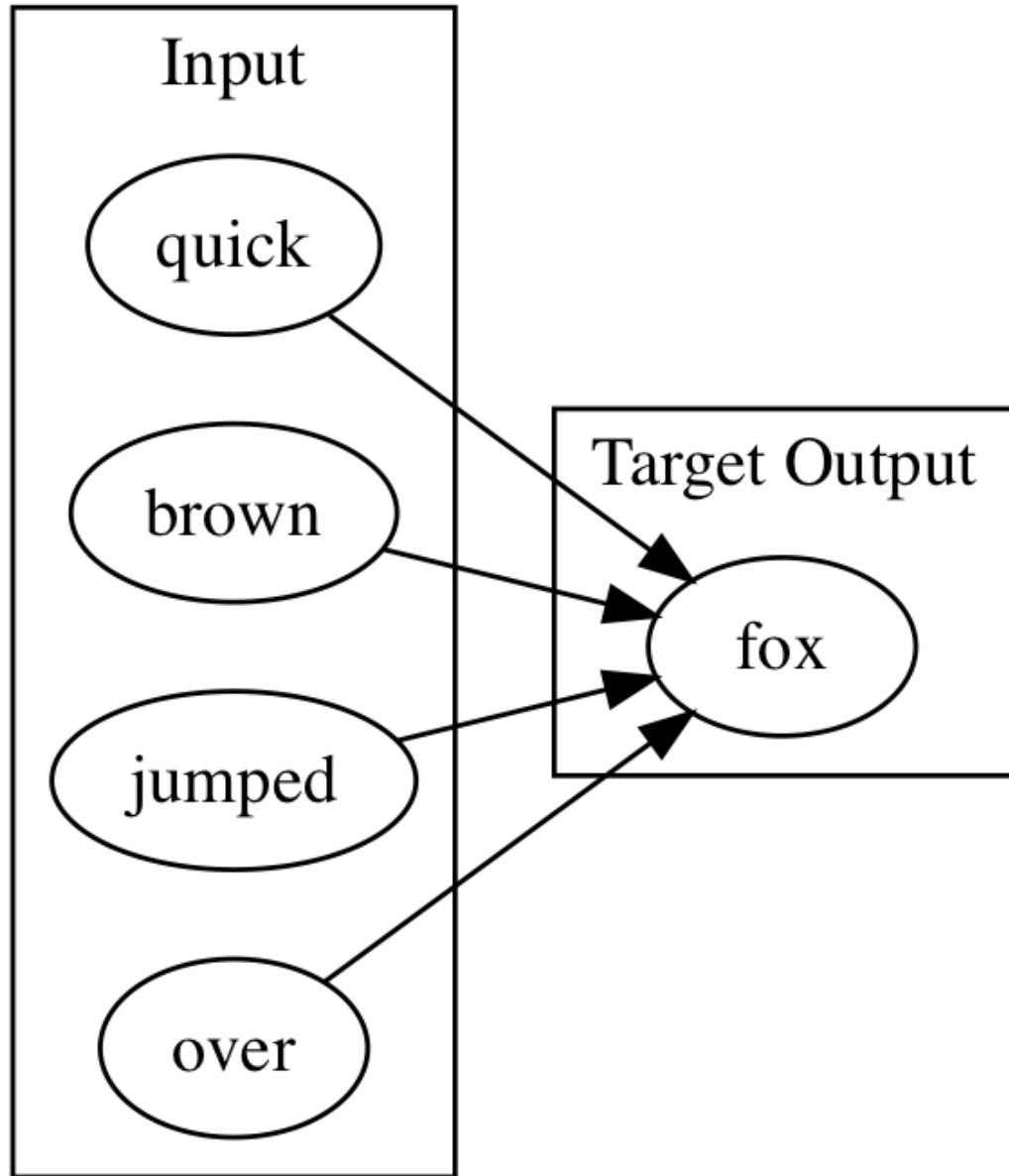
- CBOW
 - w_1, w_2, w_4, w_5 as input to neural network
 - Context words
 - Train net with w_3 as target
 - Focus word
- SKIP GRAM
 - Reverse of CBOW
 - Input is focus word
 - Output is context words

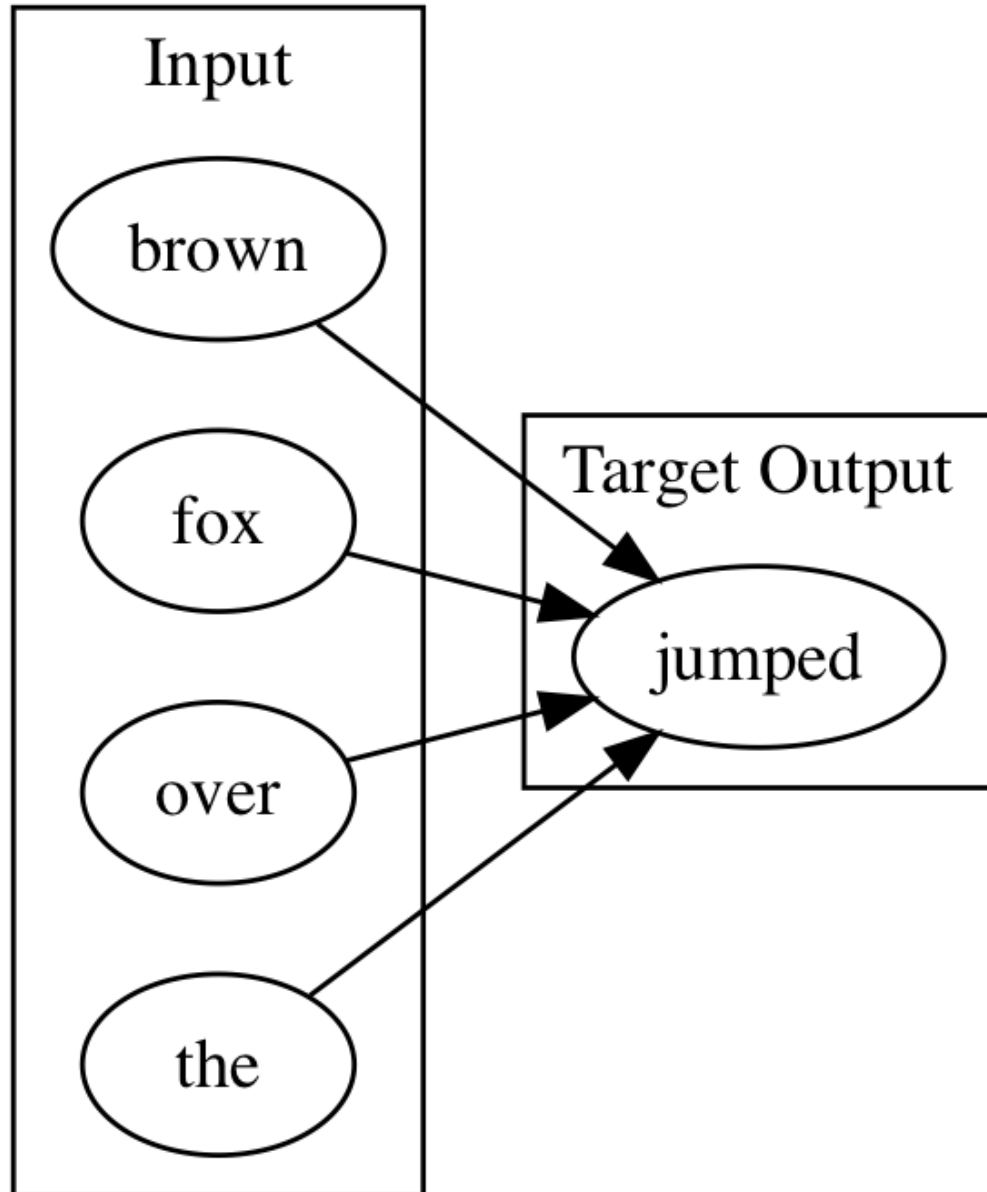


CBOW visually



CBOW visually





CBOW visually

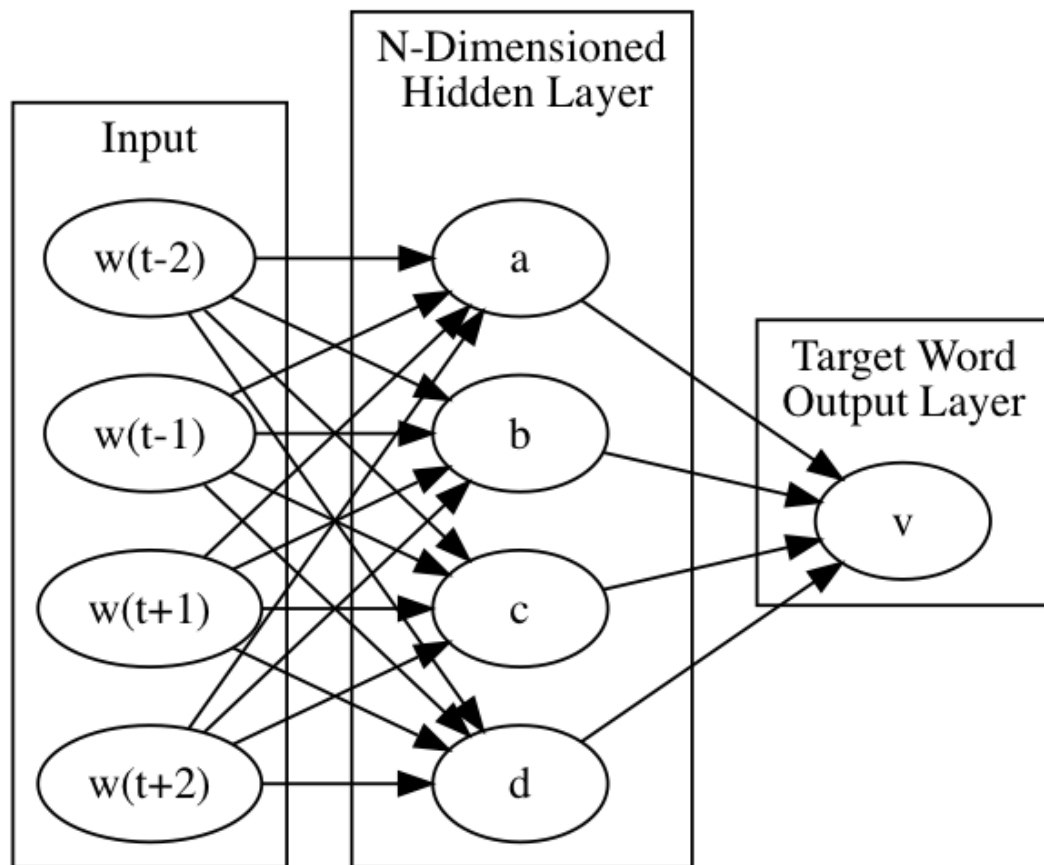


Table of Contents

- PreProcessing and tokenization
- Bag of Words
- N-Grams
- Word2Vec
- \Rightarrow Paragraph Vectors
- GloVE
- Words as Sequence of Characters



Paragraph Vectors/doc2Vec

- Extension to Word2Vec
 - Word2Vec associates words with words
 - doc2vec has additional label
 - Useful for sentiment analysis



Table of Contents

- PreProcessing and tokenization
- Bag of Words
- N-Grams
- Word2Vec
- Paragraph Vectors
- \Rightarrow GloVE
- Words as Sequence of Characters



GloVE

- Vector Representation of words obtained from unsupervised word-word co-occurrence
- Pretrained vectors available
 - Wikipedia
 - Twitter



- PreProcessing and tokenization
- Bag of Words
- N-Grams
- Word2Vec
- Paragraph Vectors
- GloVE
- ⇒ Words as Sequence of Characters



Text as Sequence of Characters

Text can be treated as sequence of characters; neural networks can be trained to answer the question "Given input character X predict the next character"; then repeat.



Character vs Word as Unit of Analysis

- How many words are there?
- How many characters are there?
- Text->word processing is hard
 - prefix, suffix, etc
 - "old school" , "New York"



Sequence of Characters as SubTree in Tree of All Character Strings

Graph of test branch teste, testi, branch tested, branch testing

In an Recurrent Neural Network(Graves LSTM) each node is a hidden state vector



Using Recurrent Neural Networks to Write Weather Forecast

After the content that describes LSTM RNN in detail we will have a lab that builds a neural network to generate characters one character at a time from a learned corpus.

In the lab we will train the network weather forecasts.

***Instructor note, foreshadow the lab , do not start the lab yet, next chapter ***



Introduction to Recurrent Neural Networks

- Overview
- Benefits
- Modeling Sequences
- Details
- BackPropagation through time
- Tuning Guidelines
- PhysioNet Example

Table of Contents

- ⇒ Overview
- Benefits
- Modeling Sequences
- Details
- BackPropagation through time
- Tuning Guidelines
- PhysioNet Example



What is a Recurrent Neural Network?

- FeedForward Network with hidden state
- Hidden state with own internal dynamics
- Information can be stored in "hidden state" for a long time

In this section when we refer to RNN we mean Graves LSTM as defined here
<https://arxiv.org/abs/1308.0850>



Long Short Term Memory RNNs Uses

- HandWriting Recognition
- Sequence/Time Series Data
- Sequence Generation
- Sequence Classification



Long Short Term Memory RNN's

- Consider dynamic state of NN as Short term
- We want to make that last a long time(Improve upon Vanilla RNN)
- Create modules that allow information to
 - Gate in
 - Gate out when needed
 - In between gate is closed and information is preserved
 - Forget Gate



Long Short Term Memory RNN's

- Logistic write gate with each node in Recurrent Layer
- Write Gate
 - ON/OFF state determined by rest of the network
 - Write State=ON
 - State is saved
- Keep Gate
 - On/OFF state determined by rest of Net
 - Keep Gate=ON
 - State is maintained
- Read Gate
 - On/Off State Determined by rest of network
 - Read Gate ON
 - Data in cell is output to network
- Data in the memory cell is actually analog
- It writes the data back to itself at each step while "keep" is closed using Weight of 1



Why RNN and Not deep FeedForward Network?

- Vanishing Gradient Challenge with Deep FeedForward Networks
- Makes training deep networks challenging
- Backward pass is Linear
 - Compound multiplication of values close to zero tend to vanish
 - Vanishing Gradient
 - Compound multiplication of large values tend to explode
- Forward pass is Non-linear
 - Activation functions (squashing functions) prevent activity vectors from exploding
- Solution is LSTM to preserve time based info isolated from backpropagation



- Overview
- \Rightarrow Benefits
- Modeling Sequences
- Details
- BackPropagation through time
- Tuning Guidelines
- PhysioNet Example



RNNs power

- Distributed Hidden State
- Multiple hidden units can be active at once
 - Can "remember" several different things
- Nonlinear
 - Allows updates to hidden state in complicated ways
- "With enough neurons and enough time they can compute anything that can be computed on your computer" (Hinton Lecture)



ARi's thoughts on content

- Condense key concepts
- When to use
- Why to use
- Additional considerations
 - BPTT
 - Activation considerations
 - Train Test split considerations



Another Benefit of RNN over FeedForward Network

- FeedForward network
 - One to one relationship input to output
- Recurrent network
 - one to many
 - One Image to many words in Caption
 - Many to many
 - English to French
 - Many to one
 - Voice classification



Benefit of Recurrent Neural Network Over FeedForward Network with Fixed Time Window

- Pre-configured window of time steps
 - Brittle hard coded
 - Requires domain knowledge of feature dependencies
- Flexible state information for flexible length events
 - Able to learn over long flexible event windows
 - Learns feature dependencies over learned flexible time window

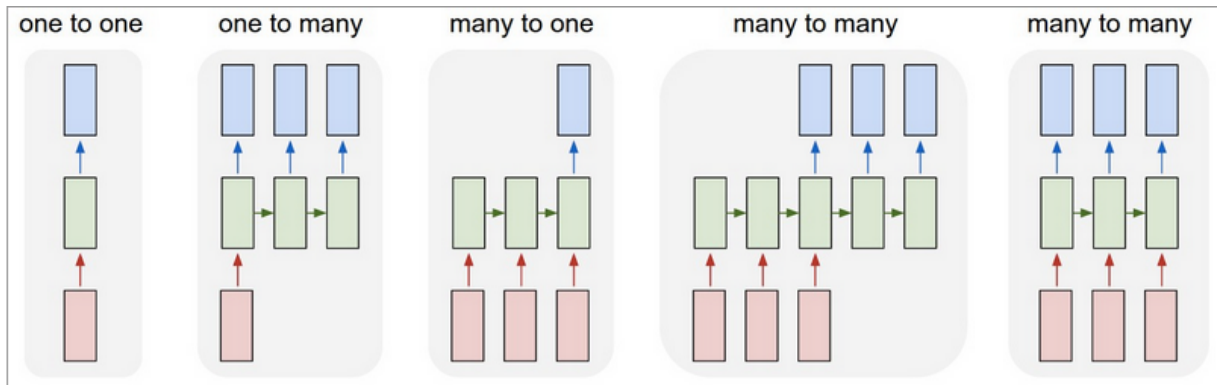


What Types of Activity can an RNN model

- Oscillation
 - motor control, walking robots
- Sequences
 - Including long term dependencies
- Text as sequences of characters
- Text as Sequences of words



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

LSTM Recurrent Neural Networks Successes

- Anomaly detection
- Handwriting Recognition
- Speech Recognition
- Image Captioning



Cursive handwriting recognition

- Input is sequence of pen coordinates as text is written
- Output is sequence of characters
 - Graves & Schmidhuber (2009)
- If sample is not live sequence of small image samples as input works



Training Data Requirements for Natural Language Processing

- RNN's require much less training data than other solutions



- Overview
- Benefits
- \Rightarrow Modeling Sequences
- Details
- BackPropagation through time
- Tuning Guidelines
- PhysioNet Example



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



Recurrent Neural Networks and Sequence Data

- Recurrent Neural Networks have the capacity to recognize dependencies in time series data
- Breaking a text corpus into a series of single characters allows the network to learn dependencies such as the most common letter after a "Q" is a "U", when a quote has been opened it should eventually be closed
- In the Lab you will train a neural network to write weather forecasts



Differences between RNN and FeedForward Networks

- RNN allows for modeling change in Vectors over time
- RNN takes Multiple sets of vectors and inputs
- FFN takes single input feature vector



Modeling Sequences

- Input Sequence to Output Sequence
 - French to Spanish
 - Speech Recognition
 - Sound Pressures to word identities
- Training Sequence try to predict the value for current step + 1



Sequence thinking outside the box

- Sequence of words make sense conceptually
- Sequence of network requests fit pattern as well
- Sequence of pixels in image?



Training Goal, one Sequence to another Sequence

- When modeling Sequential data we often want to turn one sequence into another sequence
- A phrase in english to a phrase in Spanish
- Sequence of audio ad convert into word identitites



Training Goal

- Next timestep of current sequence



Non Sequence data as Sequence data

- Pixels in an image , or Grid of pixels applied to next Grid
- works quite well, feels less natural



Supervised vs unsupervised

- Training to predict next term in sequence blurs the line between supervised and unsupervised



Patterns that may use the Long Term memory of RNN's

- Character Sequence
 - Parenthesis, quotes, brackets opened or closed
 - Relationship of period space Capitalization for beginning of sentence
- Oscillation
 - Normal
 - Abnormal
- Network activity patterns
 - Input packet followed by stream of output packets
 - Anomalies in that pattern
- Financial Transaction Sequences
 - Normal
 - Abnormal



- Overview
- Benefits
- Modeling Sequences
- ⇒ Details
- BackPropagation through time
- Tuning Guidelines
- PhysioNet Example



How an LSTM RNN works

- LSTMs contain information outside the normal flow of the recurrent network
- Network learns to store data there, read data from there, replace data in there



How the gates function

- Gates block or pass on information based on its strength and import, which they filter with their own sets of weights



How the gates learn

- Gates learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent



- Overview
- Benefits
- Modeling Sequences
- Details
- \Rightarrow BackPropagation through time
- Tuning Guidelines
- PhysioNet Example



RNN Updater: Back Propagation through time

- How an RNN is updated
- Advanced topic
- The recurrent net is conceptually a layered net that re-uses the same weights
- Layered feed forward network with weights constrained at each layer to be the same



RNN Updater: Back Propagation through time...

- Compute the gradients as usual
- Modify to meet the constraint (time constraint previous slide)
- Represent RNN as feed forward net with shared weights
- Forward pass builds stack of activities at each time slice
- Backward pass peels activities off that stack and computes error derivatives
 - That is why called BackPropagation through time
- After back prop for each time step constrain weights to match



Learning Process Review

- Different sets of weights filter the input for input, output and forgetting
- The forget gate is represented as a linear identity function,
 - If the gate is open, the current state of the memory cell is simply multiplied by one, to propagate forward one more time step



Table of Contents

- Overview
- Benefits
- Modeling Sequences
- Details
- BackPropagation through time
- ⇒ Tuning Guidelines
- PhysioNet Example



LSTM Hyperparameter Tuning

- Avoid Overfitting
 - Great performance on training Data
 - Bad performance on out-of-sample prediction
- Use Regularization helps:
 - L1
 - L2
 - dropout
- Larger Network, more likely to overfit
 - Avoiding trying to learn a million parameters from 10,000 examples
 - parameters > examples = trouble
 - More data is always better



LSTM Hyperparameter Tuning: Continued...

- Train over multiple epochs (complete passes through the dataset)
- Evaluate test set performance at each epoch to know when to stop (early stopping)
- The learning rate is the single most important hyperparameter
 - Tune this using [deeplearning4j-ui](#); see this [graph](#)
- In general, stacking layers can help
- For LSTMs, use the softsign (not softmax) activation function over tanh (it's faster and less prone to saturation (~0 gradients))
- Updaters: RMSProp, AdaGrad or momentum (Nesterovs) are usually good choices. AdaGrad also decays the learning rate, which can help sometimes
- Finally, remember data normalization, MSE loss function + identity activation function for regression, Xavier weight initialization



Table of Contents

- Overview
- Benefits
- Modeling Sequences
- Details
- BackPropagation through time
- Tuning Guidelines
- ⇒ PhysioNet Example



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
```



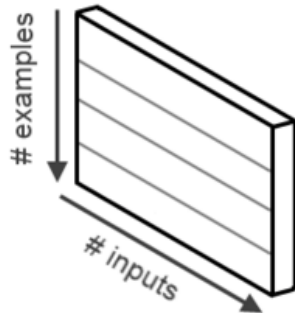
Physionet Data Continued...

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

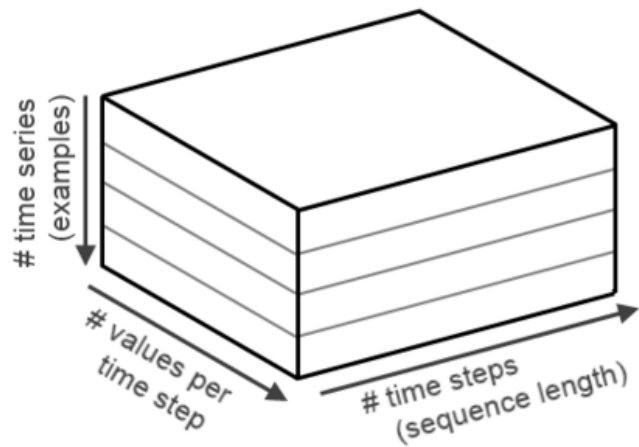


Preparing Input Data

Feed Forward Network Data



Recurrent Network Data



Preparing Input Data: continued

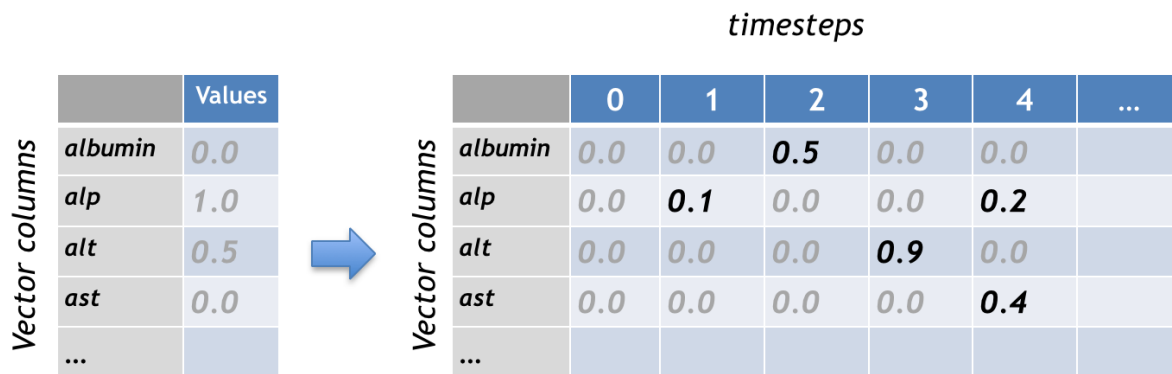
- Input was 3D Tensor (3d Matrix)
 - Mini-batch as first dimension
 - Feature Columns as second dimension
 - Timesteps as third dimension
- PhysioNet: Mini-batch size of 20, 43 columns, and 202 Timesteps
 - We have 173,720 values per Tensor input

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	...
	<i>albumin</i>	0.0	0.0	0.5	0.0	0.0	
	<i>alp</i>	0.0	0.1	0.0	0.0	0.0	
	<i>alt</i>	0.0	0.0	0.0	0.9	0.0	
	<i>ast</i>	0.0	0.0	0.0	0.0	0.0	
	...						
Input Mask (only <u>timesteps</u>)		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



Character by Character Generation of Weather Forecasts with LSTM

- Please refer to you Lab manual



Convolutional Neural Networks

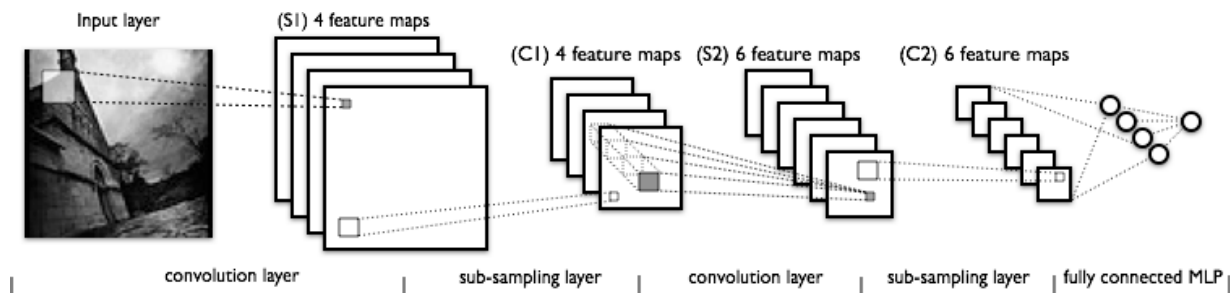
Convolutional Neural Networks

- Convolutional Neural Networks are best-in-class today for image classification
- Uses locally connected filters to “scan” for features in image data
 - Better able to deal with scale and rotation of image features
- Typically is a repeating pattern of:
 - Convolution Layer
 - ReLU Layer
 - Pooling Layer



Convolutional Neural Network

Note mention recent move to less Pool Layers

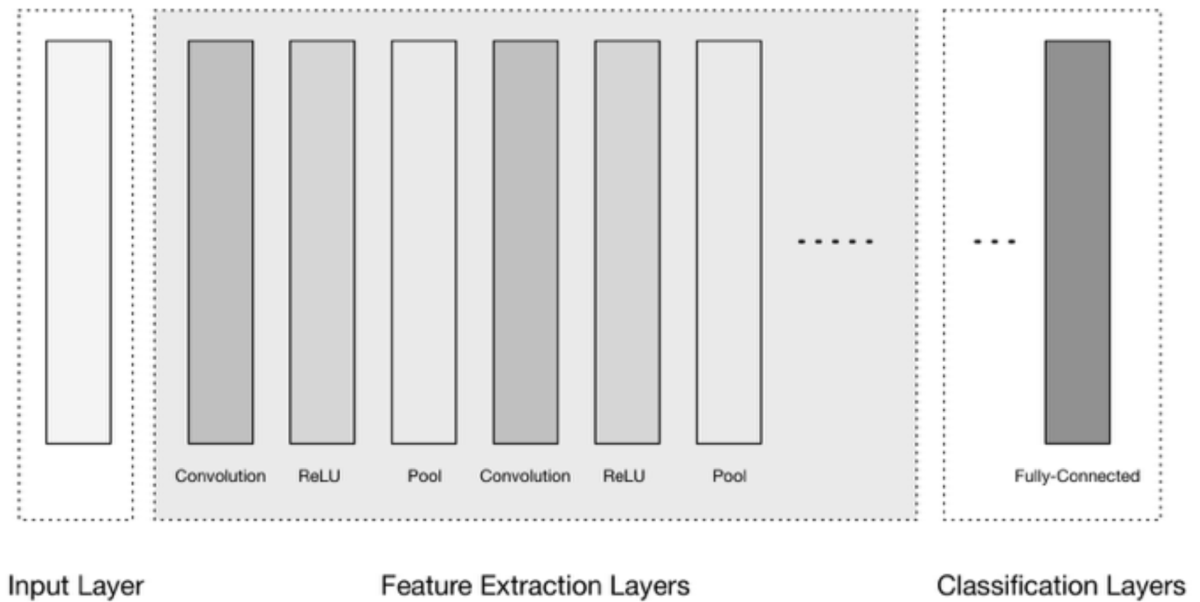


Convolutional Image Explained

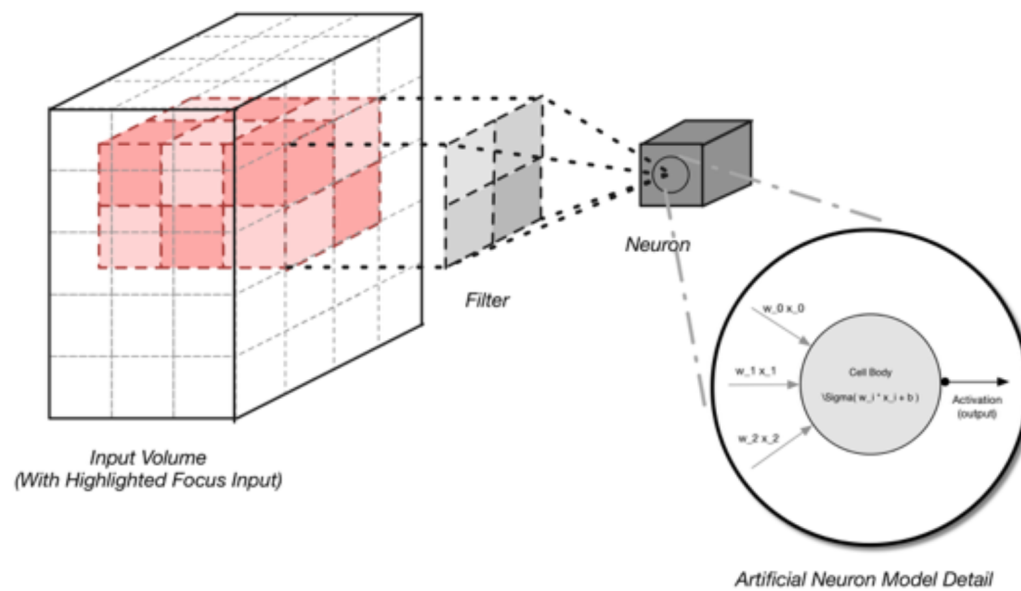
- Convolution layers “filter” x to extract features
 - Filters exploit (spatially) local regularities while preserving spatial relationships
- Subsampling (pooling) layers combine local information, reduce resolution
 - pooling gives translational invariance (i.e., classifier robust to shifts in x)
- Predict y from x with local structure (e.g., images, short time series)
 - 2D: classify images of, e.g., cats, cat may appear in different locations
 - 1D: diagnose patients from lab time series, symptoms at different times
- Special case: fully convolutional network with no MLP at “top” (filter for variable sized x 's)



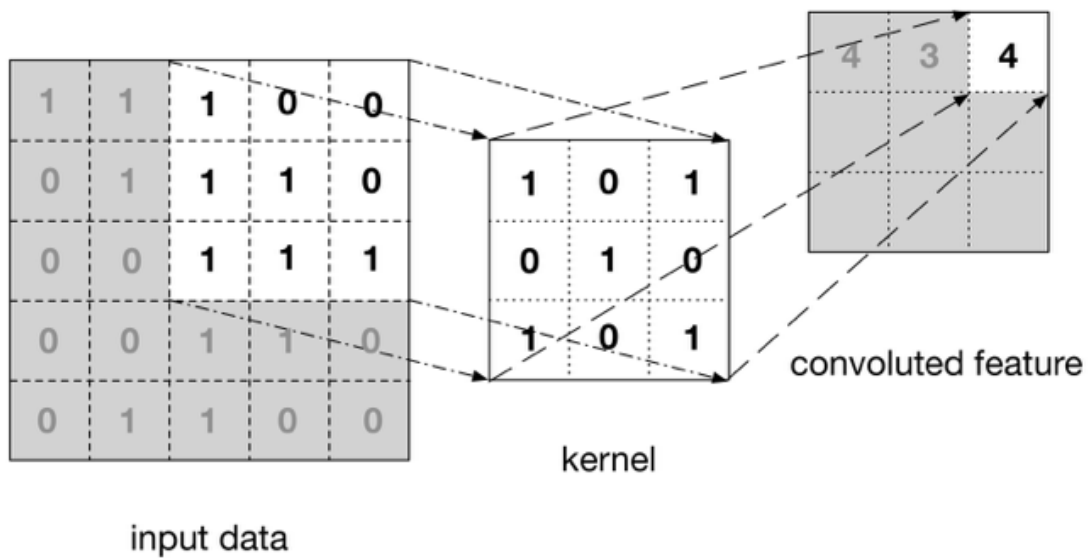
Convolutional Neural Network



3 Dimensional Input



Convolutions



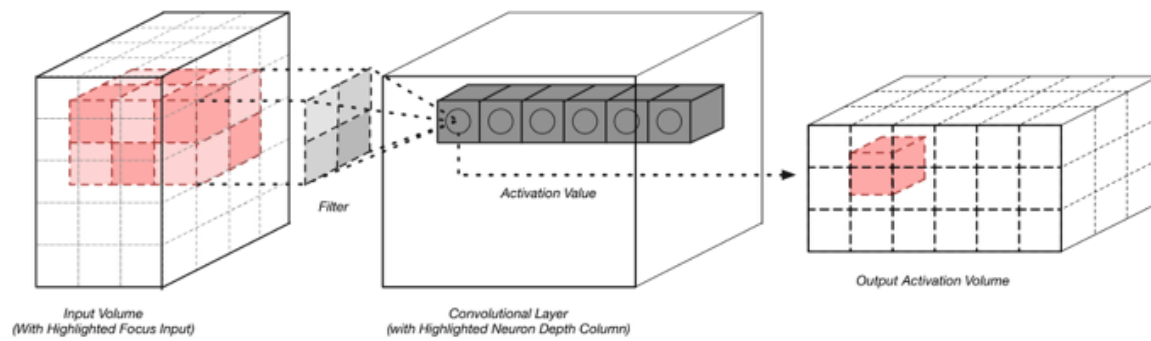
Mnist Digits



Learning Filters



Building 3D Output



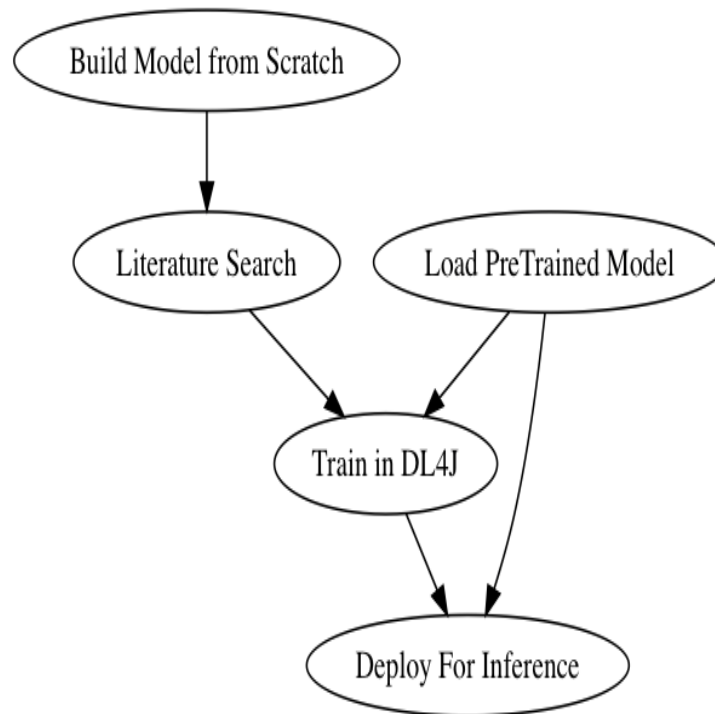
Convolutional Neural Network Lab

See your Lab Manual for Instructions

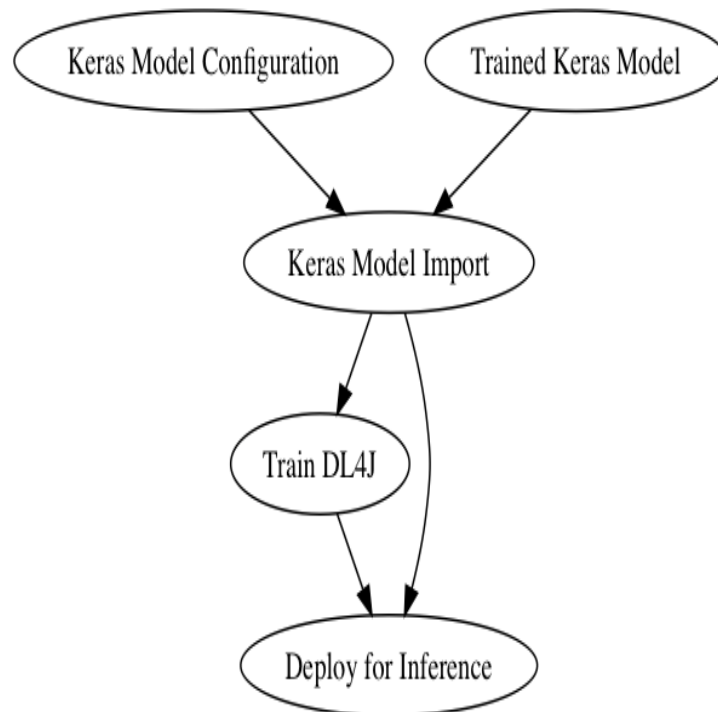


Paths to Production

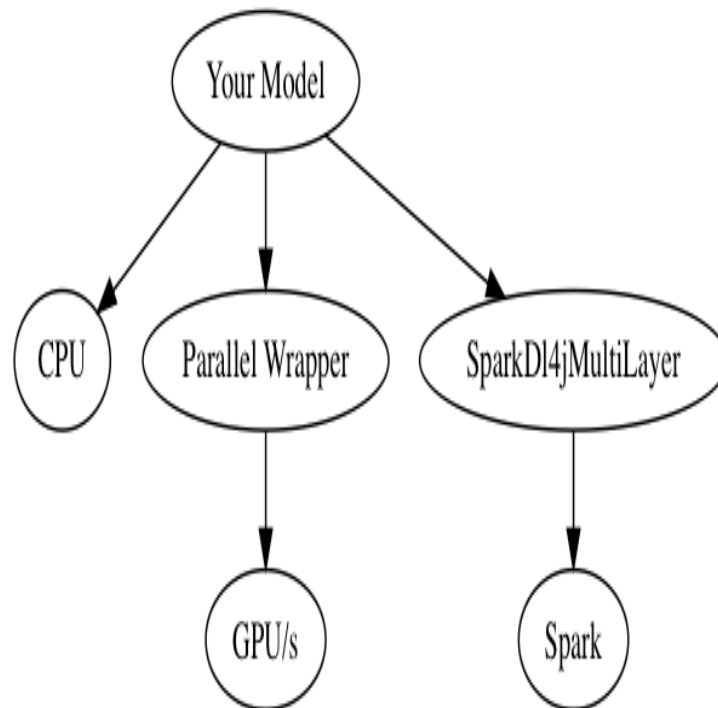
With DL4J



From Keras



Execution



Saving Keras Models



Saving just the Configuration

- `model.to_json`

```
json_string = model.to_json()
text_file = open("/tmp/iris_model_json", "w")
text_file.write(json_string)
text_file.close()
```



Saving Just the Weights

- `model.save_weights`

```
model.save_weights('/tmp/iris_model_weights')
```



Saving Weights and Configuration

- `model.save`

```
model.save('/tmp/full_iris_model')
```



Loading Keras Models into DL4J



Keras Model Import

- New Feature since 0.7.1
- Current DL4J Release 0.8.0
 - Use 0.8.0 or above



Using Model Import

- Add this to your pom.xml

```
<dependency>  
  <groupId>org.deeplearning4j</groupId>  
  <artifactId>deeplearning4j-modelimport</artifactId>  
  <version>${dl4j.version}</version>  
</dependency>
```



Computation Graph or MultiLayerNetwork

- Keras Sequential Model => DL4J MultiLayerNetwork
- Keras Functional API => DL4J ComputationGraph



Import Configuration Only

- Sequential Model Configuration import, saved in Keras with `model.to_json()`

```
CopyMultiLayerNetworkConfiguration modelConfig =  
KerasModelImport.importKerasSequentialConfiguration  
("PATH TO YOUR JSON FILE)
```

- ComputationGraph Configuration import, saved in Keras with `model.to_json()`

```
ComputationGraphConfiguration computationGraphConfig =  
KerasModelImport.importKerasModelConfiguration  
("PATH TO YOUR JSON FILE)
```



Import Configuration and Weights

- Sequential Model single file

```
MultiLayerNetwork network =  
KerasModelImport.importKerasSequentialModelAndWeights  
("PATH TO YOUR H5 FILE")
```

- Sequential Model one file for config one file for weights

```
MultiLayerNetwork network =  
KerasModelImport.importKerasSequentialModelAndWeights  
("PATH TO YOUR JSON FILE","PATH TO YOUR H5 FILE")
```



enforceTrainingConfig

- Use model only for inference?
 - `enforceTrainingConfig=false`
- Use model for further training
 - `enforceTrainingConfig=false`



Saving and Loading Trained Models

- Saving a Model
- Loading a Model
- Importing from Keras

Table of Contents

- ⇒Saving a Model
- Loading a Model
- Importing from Keras



Why Save a Model?

- Training takes time
- Deploy on multiple machines



ModelSerializer

- [JavaDoc for ModelSerializer](#)
- writeModel Method
 - Writes configuration, weights and optionally the updater to file or output stream



Table of Contents

- Saving a Model
- ⇒ Loading a Model
- Importing From Keras



Restoring a Model

- `restoreMultiLayerNetwork`
- `restoreComputationGraph`



loadUpdater

- Updater is configuration needed for further training
- Without Updater model can be used for inference only
- With Updater model can be trained further



Table of Contents

- Saving a Model
- Loading a Model
- ⇒ Importing From Keras



Keras Model Import

- Import Trained models from Keras into DL4J
- Keras Sequential -> DL4j MultiLayerNetwork
- Keras Functional API -> DL4J Computation Graph



VGG-16 Demo

- Loaded from Keras
- Saved with ModelSerializer
- Reloaded for Demo



Model Saving Lab Introduction

Main Classes Used

- ModelSerializer
 - Load and restore the model
- NativeImageLoader



Inference vs Training

- Native Image Loader in place of RecordReader
- Scales image as it is read
- Converts to INDArray



Matching Ingestion Pipeline

- Apply same scaling
- Apply same normalization
- Normalizer may depend on statistics from Training Data
- To save/restore Normalizer
 - `ModelSerializer.addNormalizerToModel`
 - `ModelSerializer.restoreNormalizerFromFile`



Model Save/Restore Lab

- Please refer to your Lab Manual



Appendix

Table of DataVec RecordReaders

Name	Summary	Uses
BaseImageRecordReader	Base class for the image record reader	Image Data
CodecRecordReader	Codec record reader for parsing: H.264 (AVC) Main profile decoder MP3 decoder/encoder Apple ProRes decoder and encoder AAC encoder H264 Baseline profile encoder Matroska (MKV) demuxer and muxer MP4 (ISO BMF, QuickTime) demuxer/muxer and tools MPEG 1/2 decoder (supports interlace) MPEG PS/TS demuxer Java player applet VP8 encoder MXF demuxer Credit to jcodec for the underlying parser	Video
CollectionRecordReader	Collection record reader. Mainly used for testing.	Testing
CollectionSequenceRecordReader	Collection record reader for sequences. Mainly used for testing.	Sequence Data
ComposableRecordReader	RecordReader for each pipeline. Individual record is a concatenation of the two collections. Create a recordreader that takes recordreaders and iterates over them and concatenates them hasNext would be the & of all the recordreaders concatenation would be next & addAll on the collection return one record	Merged data
CSVNLinesSequenceRecordReader	A CSV Sequence record reader where:(a) all time series are in a single file(b) each time series is of the same length (specified in constructor)(c) no delimiter is used between time seriesFor example, with nLinesPerSequence=10, lines 0 to 9 are the first time series, 10 to 19 are the second, and so on.	Tabular Sequence Data
CSVRecordReader	Simple csv record reader	Tabular Data
CSVSequenceRecordReader	CSV Sequence Record Reader This reader is indended to read sequences of data in CSV format, where each sequence is defined in its own file (and there are multiple files) Each line in the file represents one time step	Tabular Sequence Data
FileRecordReader	File reader/writer	Files
ImageRecordReader	Image record reader. Reads a local file system and parses images of a given height and width. All images are rescaled and converted to the given height, width, and number of channels. Also appends the label if specified (one of k encoding based on the directory structure where each subdir of the root is an indexed label)	Image Data
JacksonRecordReader	Support for JSON, XML and YAML: one record per file only, via Jackson ObjectMapper:	JSON,XML, YAML
LibSvmRecordReader	Record Reader for SVM(Support Vector Machines) content.	LibSVM content
LineRecordReader	Reads files line by line	Text
ListStringRecordReader	Iterates through a list of strings return a record. Only accepts an @link {ListStringInputSplit} as input.	Text
MatlabRecordReader	Matlab record reader	Matlab
RegexLineRecordReader	RegexLineRecordReader: Read a file, one line at a time, and split it into fields using a regex. Specifically, we are using Pattern and Matcher.To load an entire file using a Example: Data in format "2016-01-01 23:59:59.001 1 DEBUG First entry message!"using regex String "(\\d{4}-\\d{2}-\\d{2})\\.\\d{2}:\\d{2}:\\d{3}) (\\d+) ([A-Z]+) (.*)"would be split into 4 Text writables: ["2016-01-01 23:59:59.001", "1", "DEBUG", "First entry message!"]	Text with Regex

Name	Summary	Uses
RegexSequenceRecordReader	<p>RegexSequenceRecordReader: Read an entire file (as a sequence), one line at a time and split each line into fields using a regex. Specifically, we are using Pattern and Matcher to do the splitting into groups Example: Data in format "2016-01-01 23:59:59.001 1 DEBUG First entry message!" using regex String "(\\d{4}-\\d{2}-\\d{2} \\d{2}:\\d{2}:\\d{2}.\\d{3}) (\\d+) ([A-Z]+) (.*)" would be split into 4 Text writables: ["2016-01-01 23:59:59.001", "1", "DEBUG", "First entry message!"] Note: RegexSequenceRecordReader supports multiple error handling modes, via RegexSequenceRecordReader.LineErrorHandling. Invalid lines that don't match the provided regex can result in an exception (FailOnInvalid), can be skipped silently (SkipInvalid), or skip invalid but log a warning (SkipInvalidWithWarning)</p>	Text Sequence Data Regex
SequenceRecordReader	A sequence of records. sequenceRecord() is used locally. sequenceRecord(URL uri, DataInputStream dataInputStream) is used for spark etc.	Sequence Data
SVMLightRecordReader	Adapted from the weka svmLight reader June 2015 - adapted to understand HDFS-style block splits	SVMLight
TfidfRecordReader	TFIDF record reader (wraps a tfidf vectorizer for delivering labels and conforming to the record reader interface)	NLP processing, Term Frequency Inverse Document Frequency
VideoRecordReader	A video is just a moving window of pictures. It should be processed as such. This iterates over a root folder and returns a	Video
WavFileRecordReader	Wav file loader	Audio