Table of Contents

Deep Learning: Building Neural Networks Using Deeplearning4j

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

Certification

- Class prepares you for Certification Exam
- Exam voucher to take Exam

Contents

Introductions

- Intro to Skymind
    - add something here
- Instructor Intro
- Student Intro

Introductions

- Why are you here?
- What do you want to learn?
- What chapter interests you?
- what is your background?

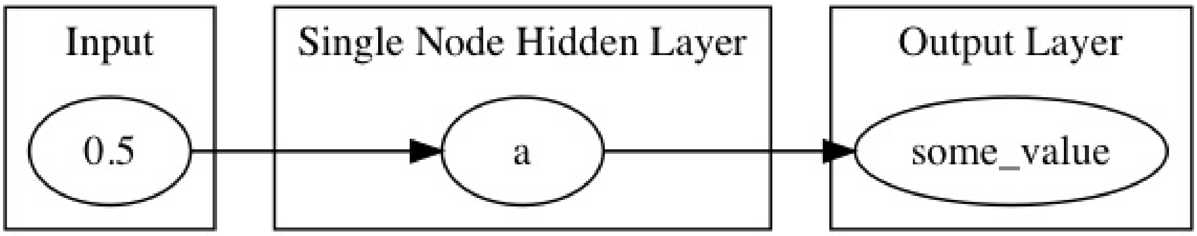
Simplest Neural Net Introduction

Table of Contents

Simplest Neural Network

- Simple Sample Network
- One input, one hidden layer with one neuron, one output
- FeedForward Network

Simplest Network



How is input handled ?

Input to a Neural Network is always Numeric. Typically a MultiDimensional Array, in this case.. not so many dimensions.

How is output Generated ?

The Output Layer of a Neural Network can be configured for different purposes.

- One of x possible classes
- True/False
- Range of Values
- and more

Simple Network demonstration and Q & A

- The instructore will demonstrate the simple Network

Settings important to all Neural Networks

- Learning Rate
    - Applies to whole Network
    - If output is 10 and expected output is 1 how big of a step to take to correnct error
- Activation Function
    - Set per layer
    - Sets the "trigger" on what output to emit
    - More on these later
- Loss Function
    - Set on Output Layer
    - How error is calculated
    - More on this later
- Updater
    - Applies to whole Network
    - How weights are updated

Simplest Network Setting

- learningRate 0.005
- updater Nesterovs variant of Stochastic Gradient Descent with Momentum
- Output Activation Identity
- Hidden Layer Activation TANH
- Number of Epochs 500

Introduction to Machine Learning

Introduction to Machine Learning

- Topics Covered

This section covers the big picture of Machine Learning and how deep learning fits in.

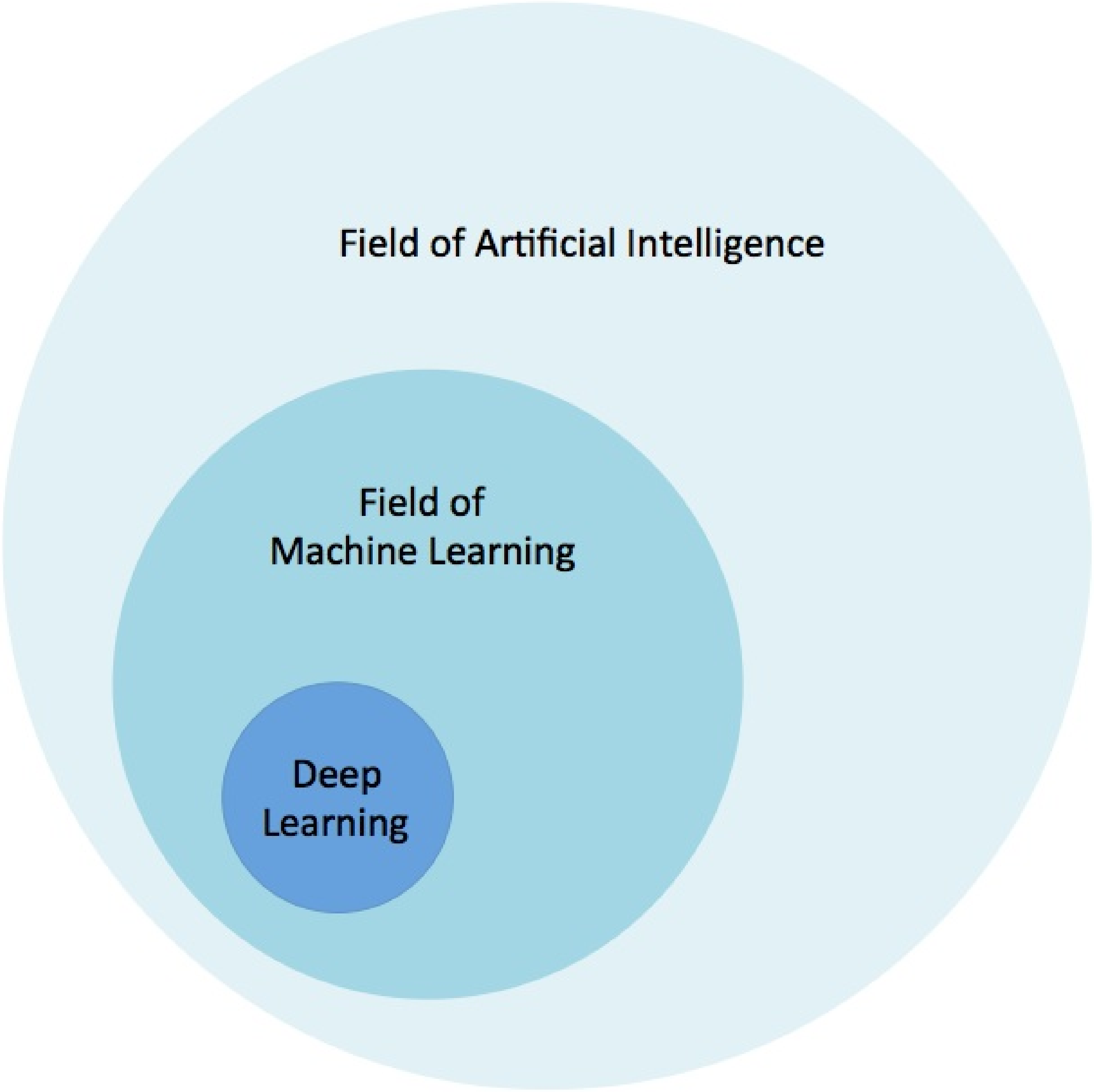
1. Machine Learning
2. Data Science / Data Mining
3. Deep Learning

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


Field of Artificial Intelligence
Field of Machine Learning
Deep Learning

Data Science / Data Mining

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

- 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

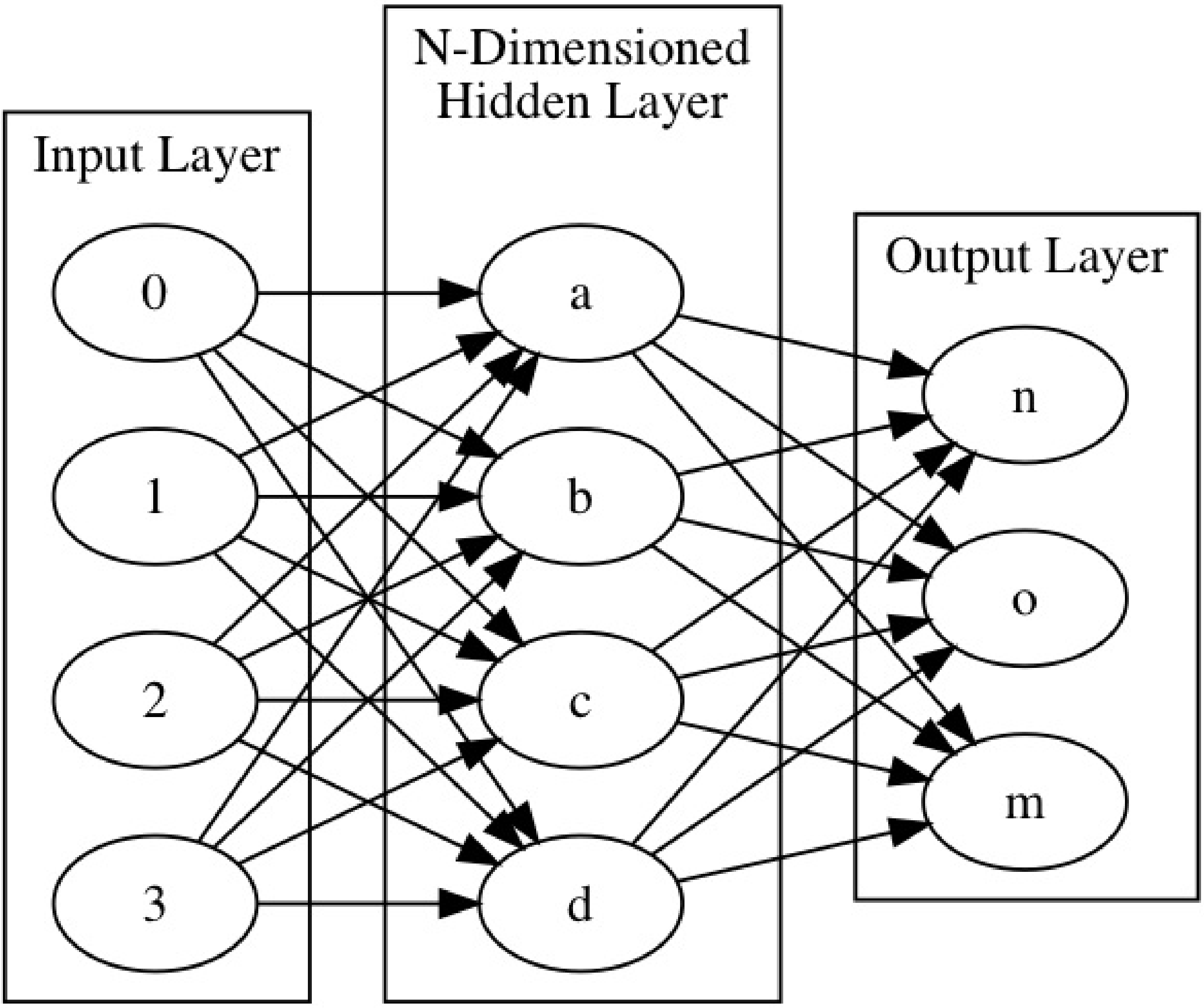
Biological Neurons

- 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.

A Neural Network



Using Neural Networks

Framing the Questions

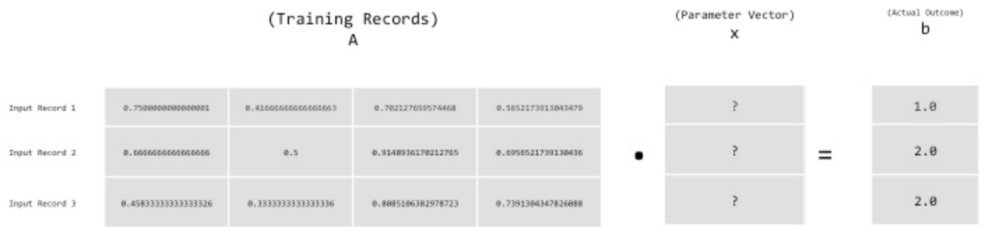
- To build models we have to define
    - What is our training data (“evidence”)?
    - What kind of model (“hypothesis”) is appropriate for this data?
    - What kind of answer (“inference”) would we like to get from the model?
- These questions frame all machine learning workflows

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	0.41666666666666663	0.702127659574468	0.5652173913043479		?		1.0
Input Record 2	0.6666666666666666	0.5	0.9148936170212765	0.6956521739130436	•	?	=	2.0
Input Record 3	0.45833333333333326	0.33333333333333336	0.8085106382978723	0.7391304347826088		?		2.0

Demo: Simplest Network

- Your Instructor will demonstrate a simple Neural Network

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
- 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 $Ax = b$ 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

Vectorization Tools

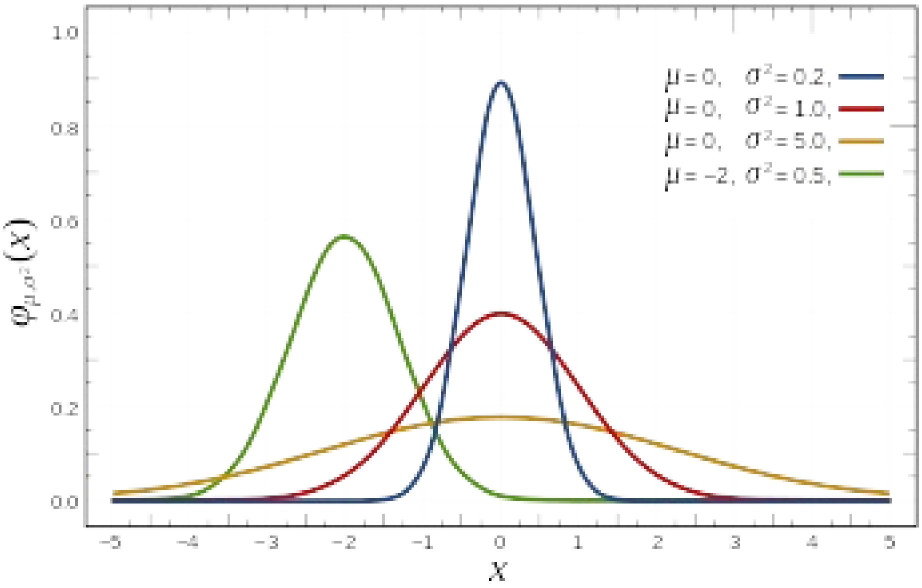
- DataVec provides tools for ingestion raw data and converting to Vectors and Matrices
    - CSV data
    - Image data
    - Sequence data
    - and more...

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

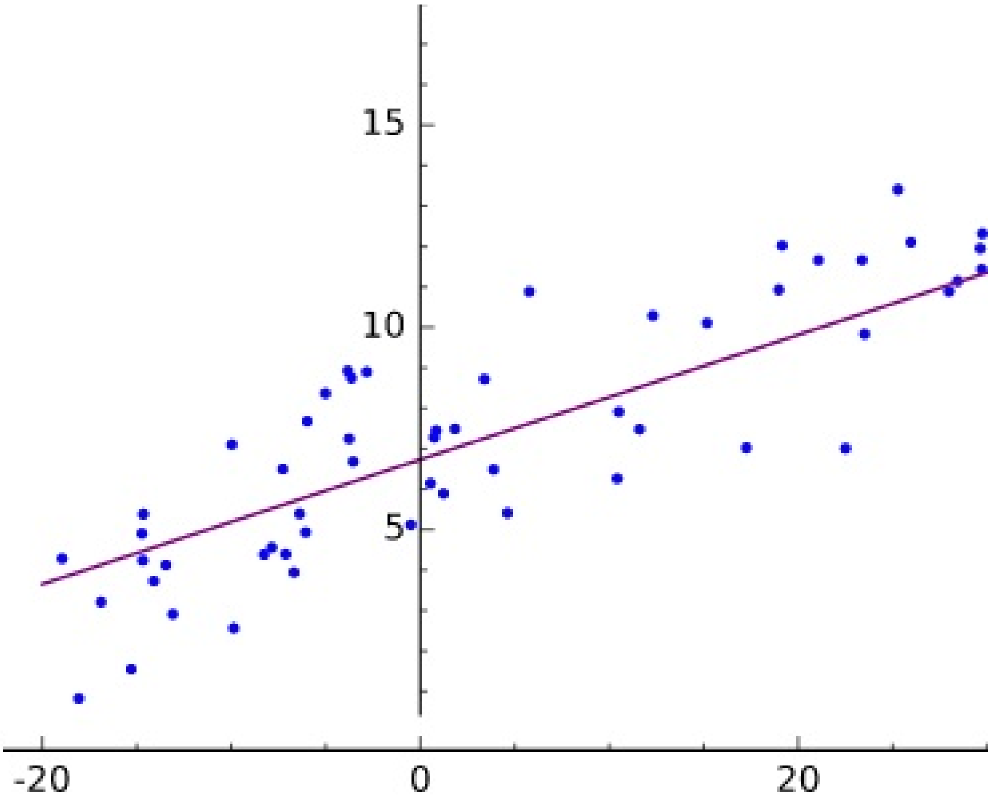
- A specification of the stochastic structure of random variables
- In statistics we rely on making assumptions about how the data is distributed
  - To make inferences about the data
- We want a formula specifying how frequent values of observations in the distribution are
  - And how values can be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

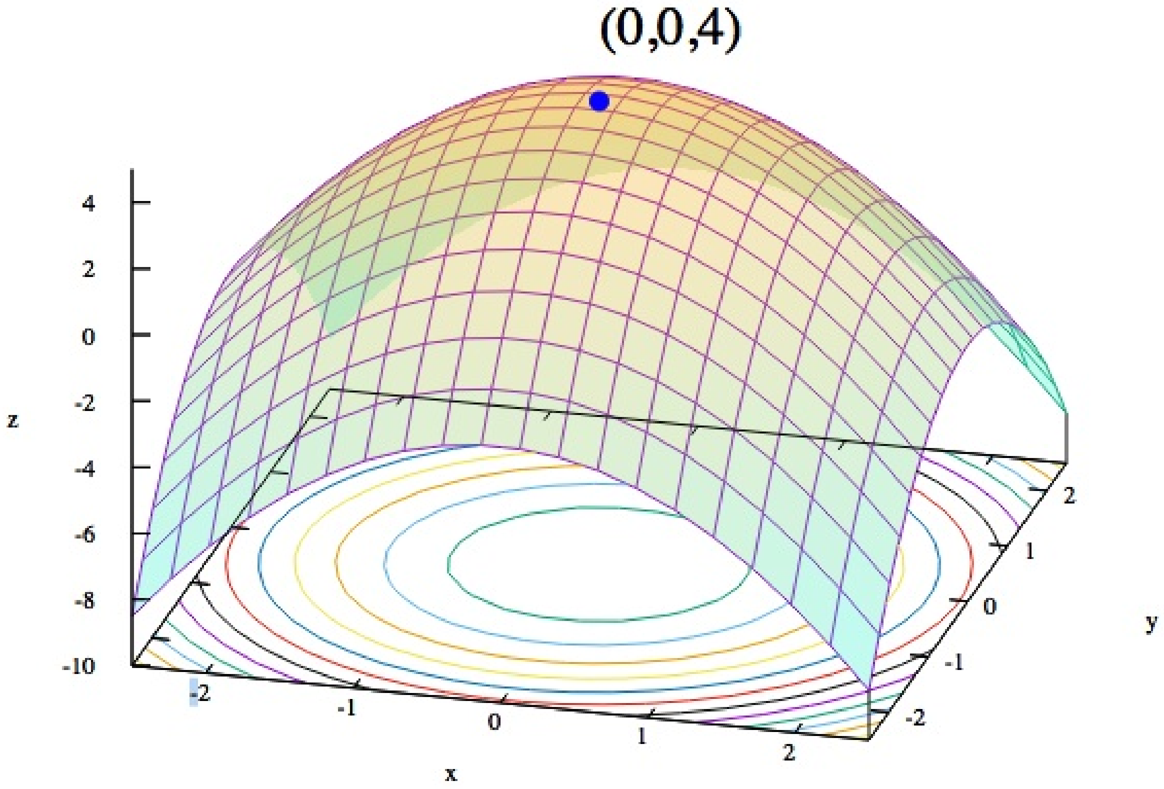
Fitting the Training Data



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

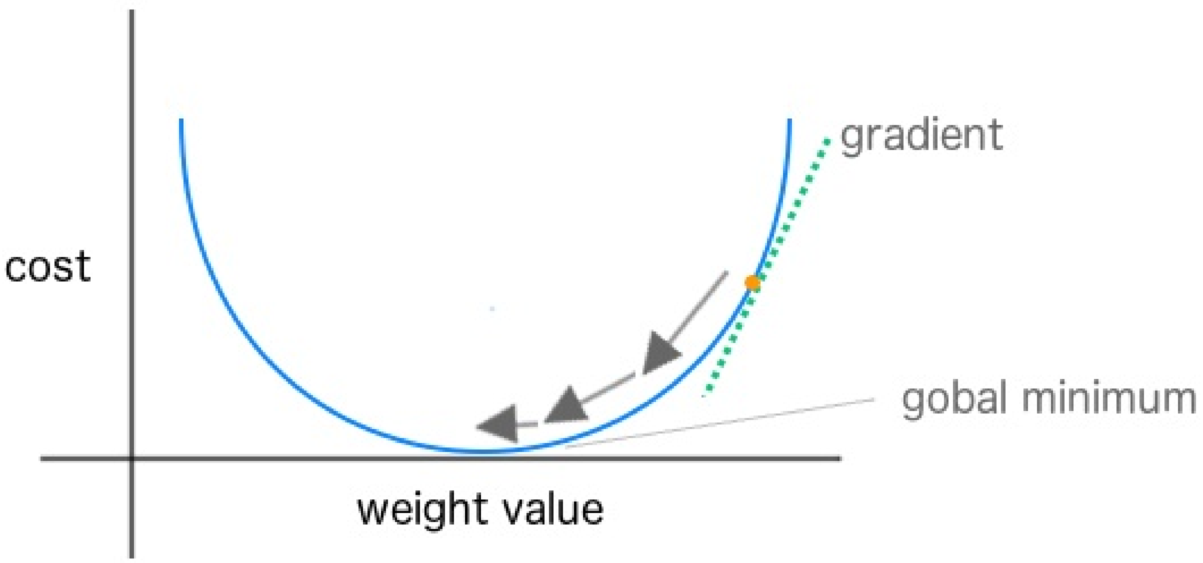
- Optimization method where we consider parameter space as
  - “hills of error”
  - Bottom of the loss curve is the most “accurate” spot for our parameter vector
- We start at one point on the curved error surface
  - Then compute a next step based on local information
- Typically we want to search in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

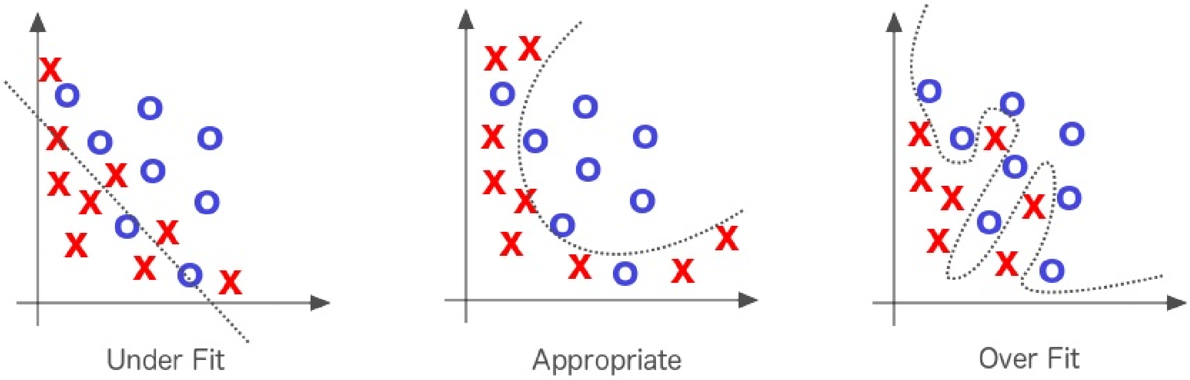
TH-clean this up

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



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 unserpervised Learning

NEED STRUCTURE PAGE HERE, SOMETHING LIKE

FRAMING THE QUESTION

- Classification
- Regression
- Clustering .....

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

Clustering

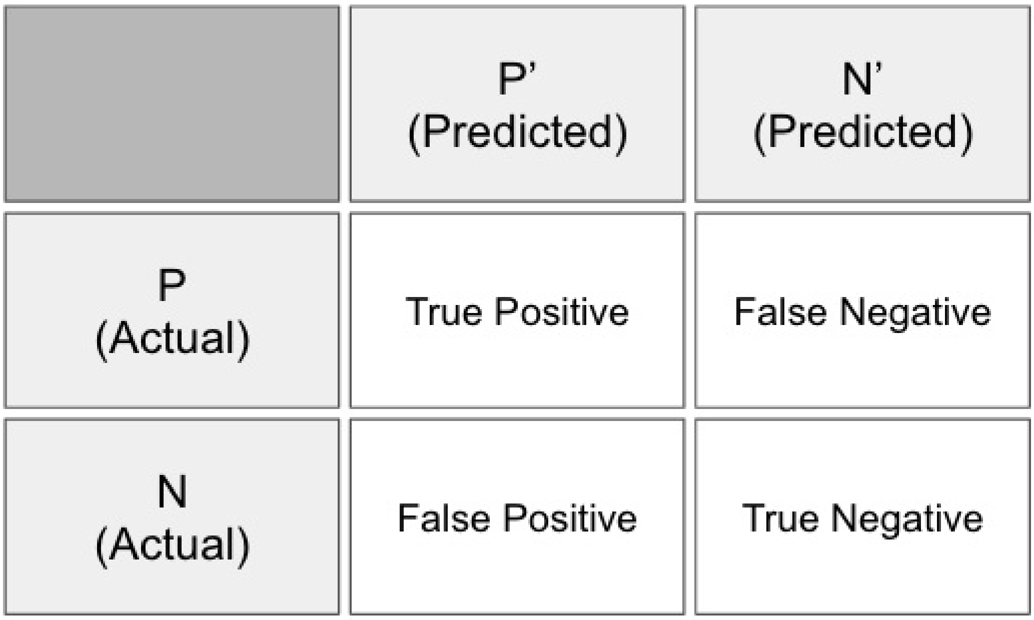
- Typically unsupervised learning
    - “K-Means Clustering”
- Example
    - “cluster K groups of similar news articles together”

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

Chapter Questions

- Is all DeepLearning Machine Learning?
- Is all Machine Learning Deep Learning?
- Need more? Probably need to structure the chapter a little bit so it answers specific questions

DeepLearning4J Overview

DeepLearning4J Overview

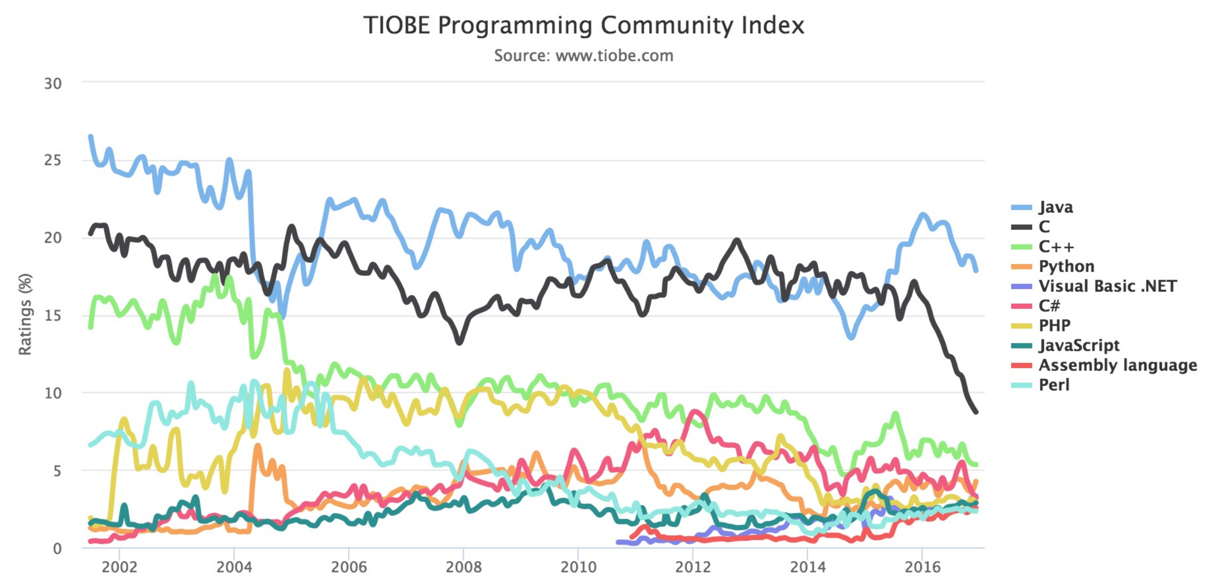
- Goals of the DeepLearning4J project
- Parts of the DeepLearning4J project
    - DataVec
    - ND4J
    - DeepLearning4J

- **Goals of the DeepLearning4J project** To provide a java based toolkit for building, training and deploying Neural Networks.
- DeepLearning4J sub-projects
  - DataVec
  - ND4J
  - DeepLearning4J

Goals of the DeepLearning4J project

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

Why Java?



TIOBE Programming Community Index

Source: www.tiobe.com

- Goals of the DeepLearning4J project
- **Parts of the DeepLearning4J project**
  - DataVec
  - ND4J
  - DeepLearning4J

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

-note- I have better content here, add from this page..
https://deeplearning4j.org/Deeplearning4J-ETL-UserGuide

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 handles 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 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. We are 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 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);

        }
```

Class Details

We could spend a whole day on DataVec, a whole day on ND4J and then a whole day on DL4j. Instead we will focus on particular network problems and teach the needed rewuirements for each, then repeat. Covering the key concepts along the way.

Chapter Questions

- DataVec Question
- ND4J question
- DL4J question

Introduction to DataVec Lab

Table of Contents

What is DataVec

- ETL(Extract Transform Load) tool for DeepLearning

Data Sources

- Text Data in CSV file
- Directory of Images
- Video
- Audio
- Sequence or Time Series Data

DataVec tools

- RecordReader
  - Iterates over a series of Records returns an iterator over an ArrayList of Writables
  - Writable = efficient serialization framework, container for Text, images, etc

CSV text data

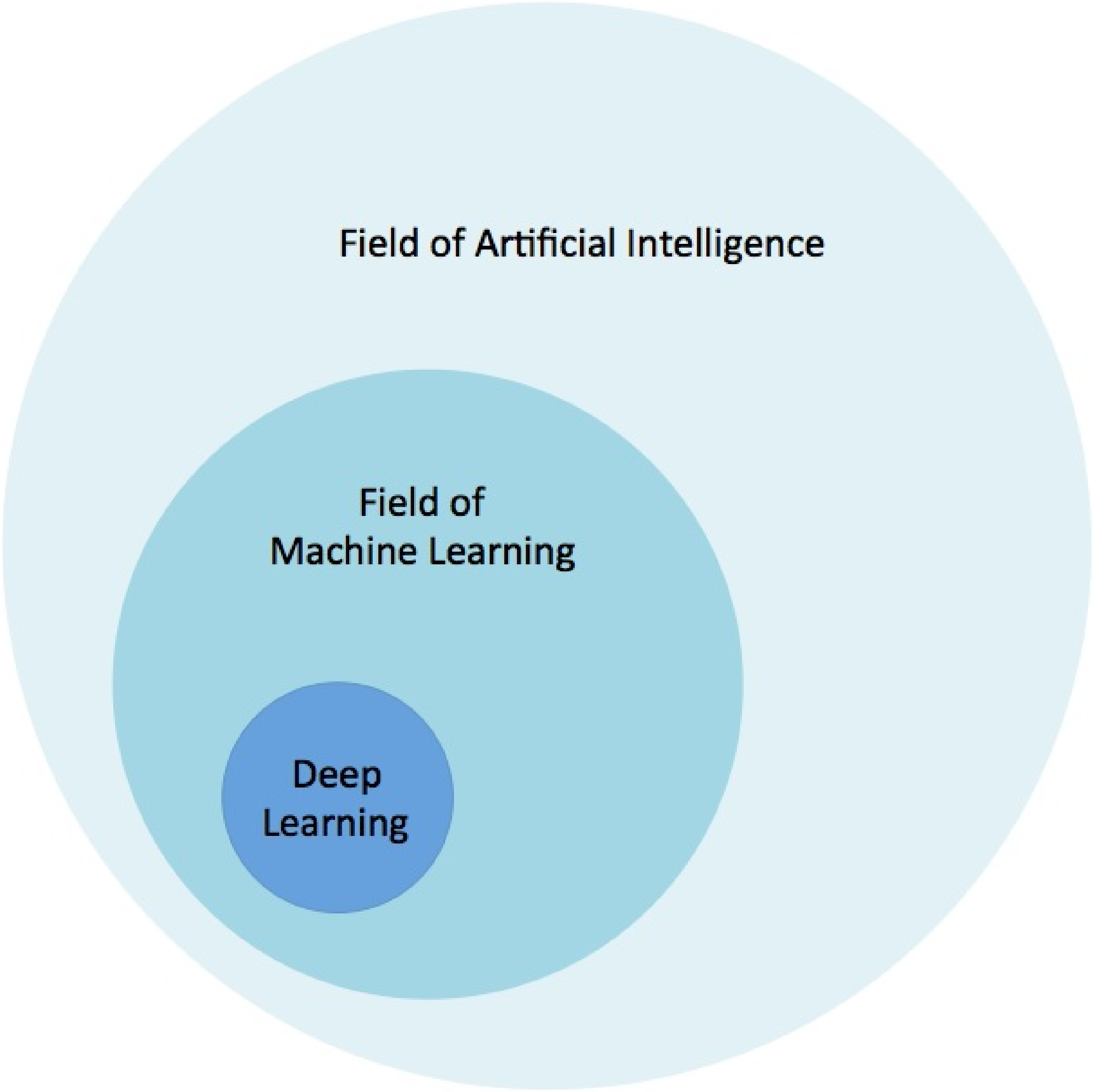
- Use CSVRecordReader
- Specify lines to skip if any
- Specify delimiter if not tab delimited
- initialize the record reader

DataSetIterator

- Takes Record Readers arrays of writables, and creates INDArray
- SPecify recordreader, what field is label, batchsize

Data Sources

A


Field of Artificial Intelligence
Field of Machine Learning
Deep Learning

Machine Learning Compared to Data Science/Mining

DeepLearning INTRO

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

Table of Contents

Defining Deep Learning

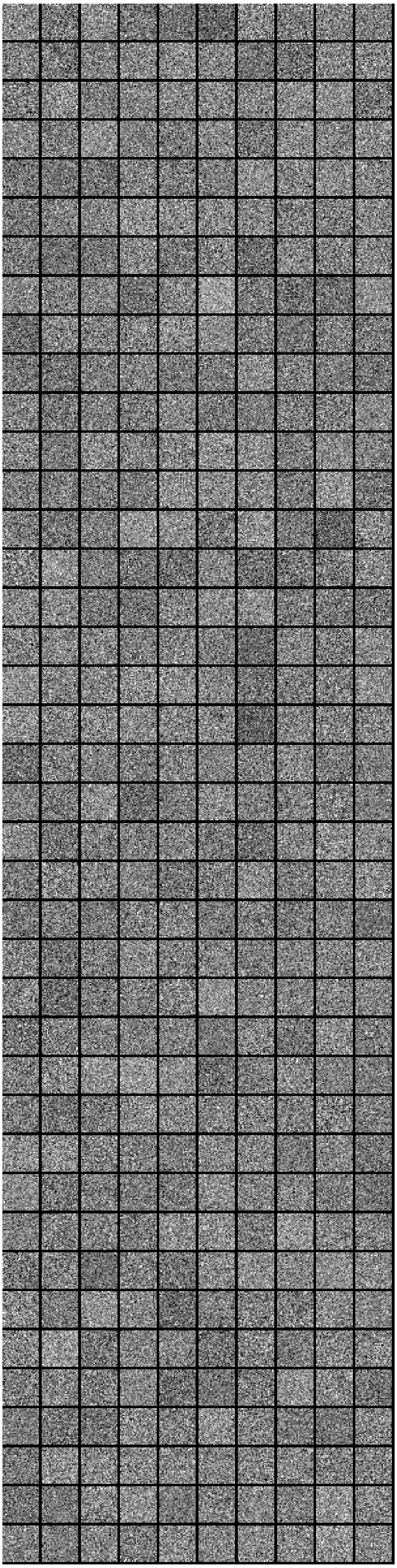
- Higher neuron counts than in previous generation neural networks
- Different and evolved ways to connect layers inside neural networks
- More computing power to train
- Automated Feature Learning

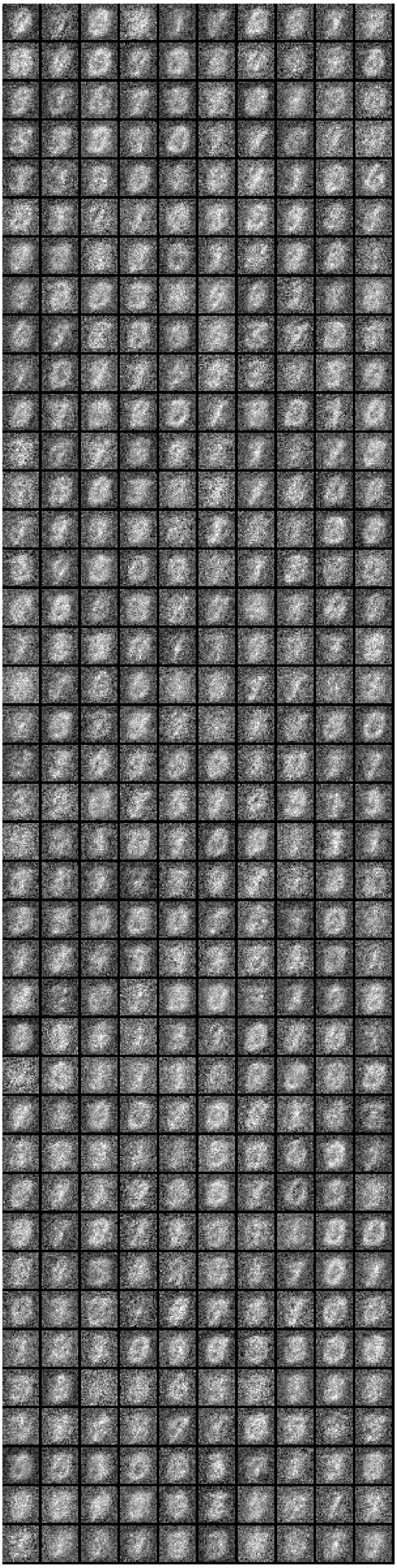
Automated Feature Learning

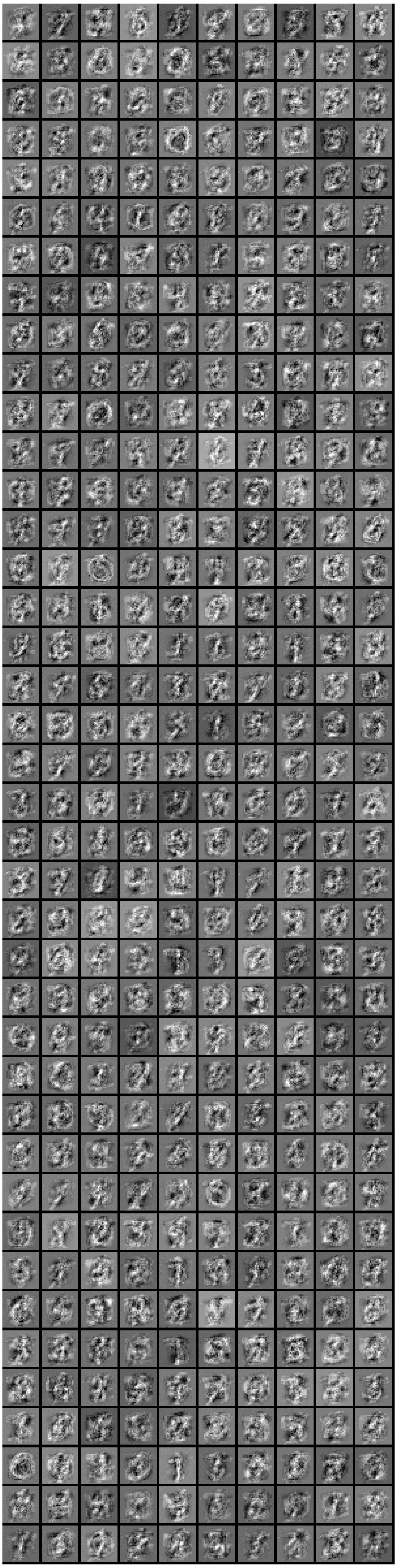
- Deep Learning can be thought of as workflows for automated feature construction
  - From “feature construction” to “feature learning”
- As Yann LeCun says:
  - “machines that learn to represent the world”

MNist Image Learning Example

- would be nice to make this a demo -
- 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

Factors influencing DeepLearning Craze

- GPU's enabling matrix-matrix computation
    - Can train extremely useful networks in weeks rather than years
- More data to traing
    - Can deliver enough data so that the network can learn needed patterns
    - Compare it to biological networks, how much data does an infant train on before first word? A LOT !!

Recent success of Deep Learning

- Image
- Speech
- other

This needs broken out and clarified, imagenet, speech, etc

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…

-- need to move this from Josh, conversational style to more formal-

- 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

Choosing the Right Architecture

- 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 Feed Forward Neural Network: 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

The fundamental pieces of a Neural Network

- Activation function, defined per layer for all neurons in that layer, the activation function determines what output signal based on input signal.
- Loss Function, how error is calculated so that network can modify weights and train
- Optimization algorithm determine how given a specific error calculation the weights are modified

Hyper Parameters Summarized

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

Activation Function

- An activation function determines what output signal is generated based on the input signals.
- Borrowed concept from biological neurons where a neuron "fires" past a certain threshold.
- Some evolution has occured over time.

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

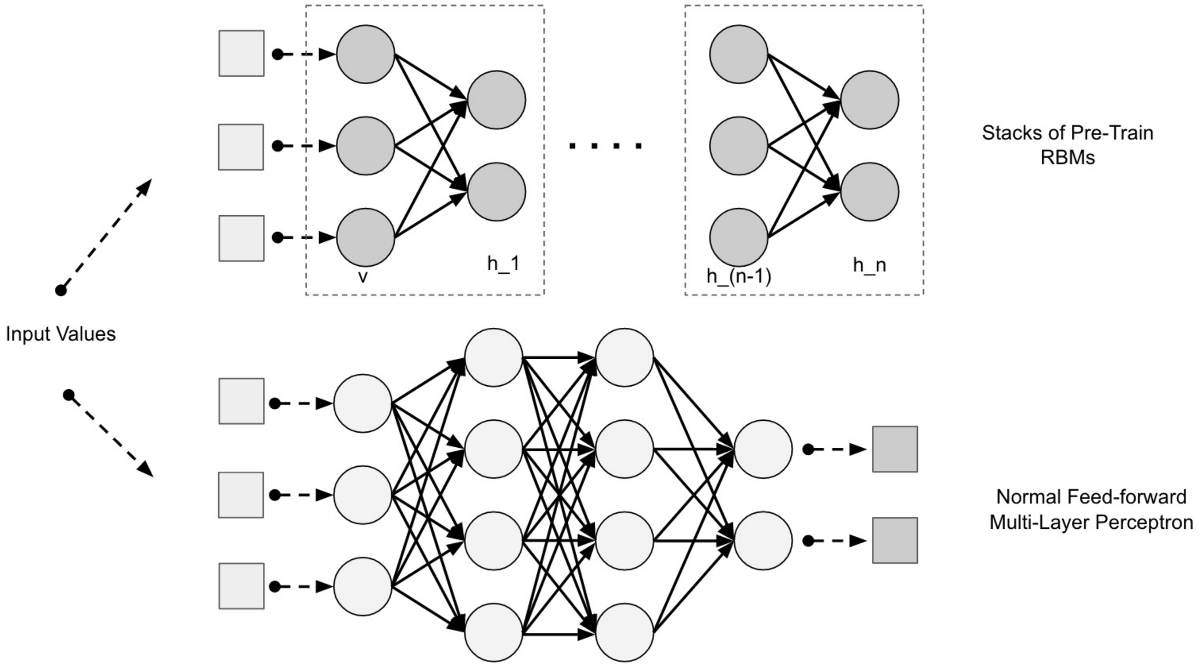
Hyperparameters

- 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



TO DO

--add pictures of Feed Forward, LSTM, etc.

Optimization Algorithms

Table of Contents

Stochastic Gradient Descent Learning

Mini Batch most widely used.

Error Surface

Horizontal axis is weights of Neural Net

Vertical is error it makes

For SINGLE Linear Neuron with squared error this is always a quadratic bowl

Vertical Cross Sections are parabolas

Horizontal Cross Sections are elipses

For MultiLayer it is much more complex, but it is smooth and locally approximation as quadratic bowl is appropriate.

Go Downhil to reduce error

- direction of steepest descent?
    - Not so good
    - tolerable but imperfect

Big Learning Rate

- Slosh to an fro across parabola

Optimization Algorithm

What we want to achieve, is that we go quickly along the ravine in directions that have small, but very consistent gradients. And we move slowly in directions with these big, but very inconsistent

-

Stochastic Gradient Descent

Choices Full Batch Less than full batch Individual case == ONLINE

Mini Batch

- If dataset is highly redundant then Half Batch is a good as full
- Less computation
- Gradient for many cases leads to matrix-matrix multiplication
- YEAH GPU's !!!
- Need to be balanced for classes
    - MMMMMM
    - FFFFFF
    - Bad *MFMFMFM = good
- SHUFFLE

Answer Use Mini Batch

-

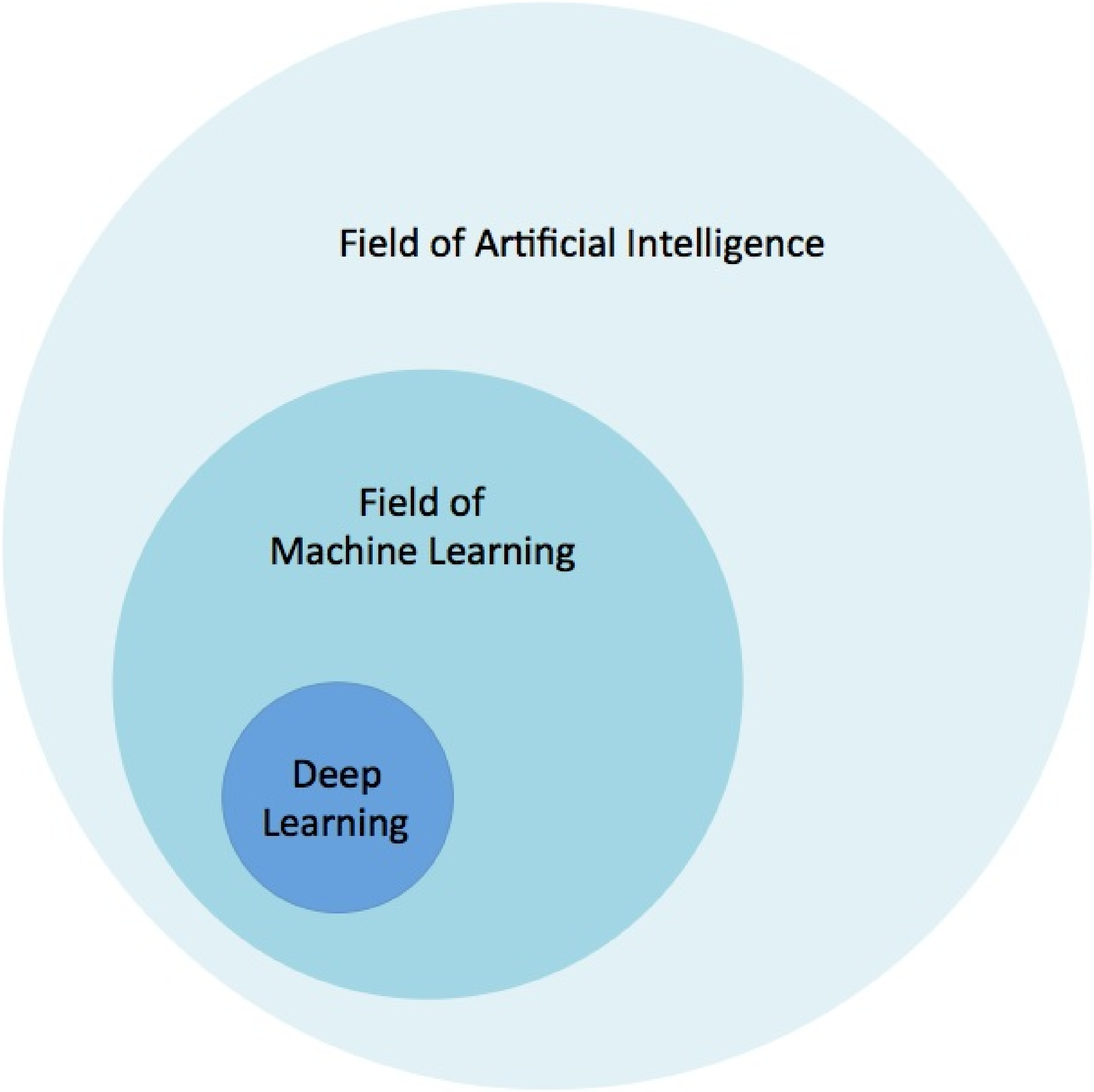
Starting steps

- Set Learning Rate
- Is error increasing or oscilating wildly?
    - Reduce the rate
- Is the error decreasing slowly
    - Increase the learning Rate

** Not sure how to adjust Learning Rate in DL4J

- Turn down the learning rate when the err

A Diagram


Field of Artificial Intelligence
Field of Machine Learning
Deep Learning

Machine Learning Compared to Data Science/Mining

Introduction to Recurrent Neural Networks

Table of Contents

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

Benefit of RNN

- Instead of pre-configured window of time steps
- Flexible state information for flexible length events

Hidden Markov models and speech recognition

Use for speech recognition in the 70's not neural net

Not sure I should mention them

Benefits of RNN's

- Distributed Hidden State
- Several different units can be active at the same time
- Can remember several different things
- Non-linear can update hidden state in complicated ways
- Hinton quote "With enough neurons and enough time, a recurring neuron network can compute anything that can be computed by your computer." coursera course

More Benefits What can RNN's model, what behavior can they exhibit

- Oscillations, motor control, walking robot
- Settle to point attractors

Challenges of RNN's

- Comp;exity makes them hard to train

Training

- Back Propogation Through time
- think of it as feed forward network with constrained weights.

Think of RNN as discrete time steps and (see Hinton Coursera)

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

Four ways to train an RNN

- LSTM Make the RNN out of little modules that are designed to remember values for a long time.

    - Use Better Optimizer HEssian Free

  That can deal with very small gradients. (Martens & Sutskever 2011)

    - Echo State Networks

  This sound cool, oscilators that reverberate for a while.

Not sure if this makes sense for this course

LSTM in depth

Consider the current state as short term memory, then we want to find a way to make this long term

Add modules that allow information to be gated in, and information to be gated out when needed.

In between information being gated in and gated out the gate is closed allowing it to remember the gated state.

->Gate open->Information stored -> Gate open information forgot->

Repeat.

LSTM in depth

- Very succesful recognizing handwriting

LSTM in depth

- Information gets into the cell whenever a logistic write gate is on

The rest of the recurrent network determines the state of that write gate, and when the rest of the recurrent network wants information to be stored, it turns the write gate on, and whatever the current input from the rest of the net to the memory cell is, gets stored in the memory cell.

-plagiarized Hinton-

The information stays in the memory cell so long as its keep gate is on. So again, the rest of the system is determining the state of a logistic keep gate, and if it keeps it on, then the information will stay there.

And finally, the information gets read from the memory cell so that it then goes off to the rest of the recurrent neural network and influences future states and it's read by turning on a read gate, Which again is a logistic unit controlled by the rest of the neural network.

The memory cell actually stores an analog value, so we can think of it as a linear neuron that has an analog value and keeps writing that value to itself at each time step by a weight of one, so the information just stays there.

Grab and create a version of his discussion about keep gates, Lecture 7 Final video.

Cursive handwriting recogntion

Input is squence of pen coordinates as text is written

Output is sequence of characters

Graves & Schmidhuber (2009)

If timing of pen is not known, the sequence of small images as input works

For Character Recognition

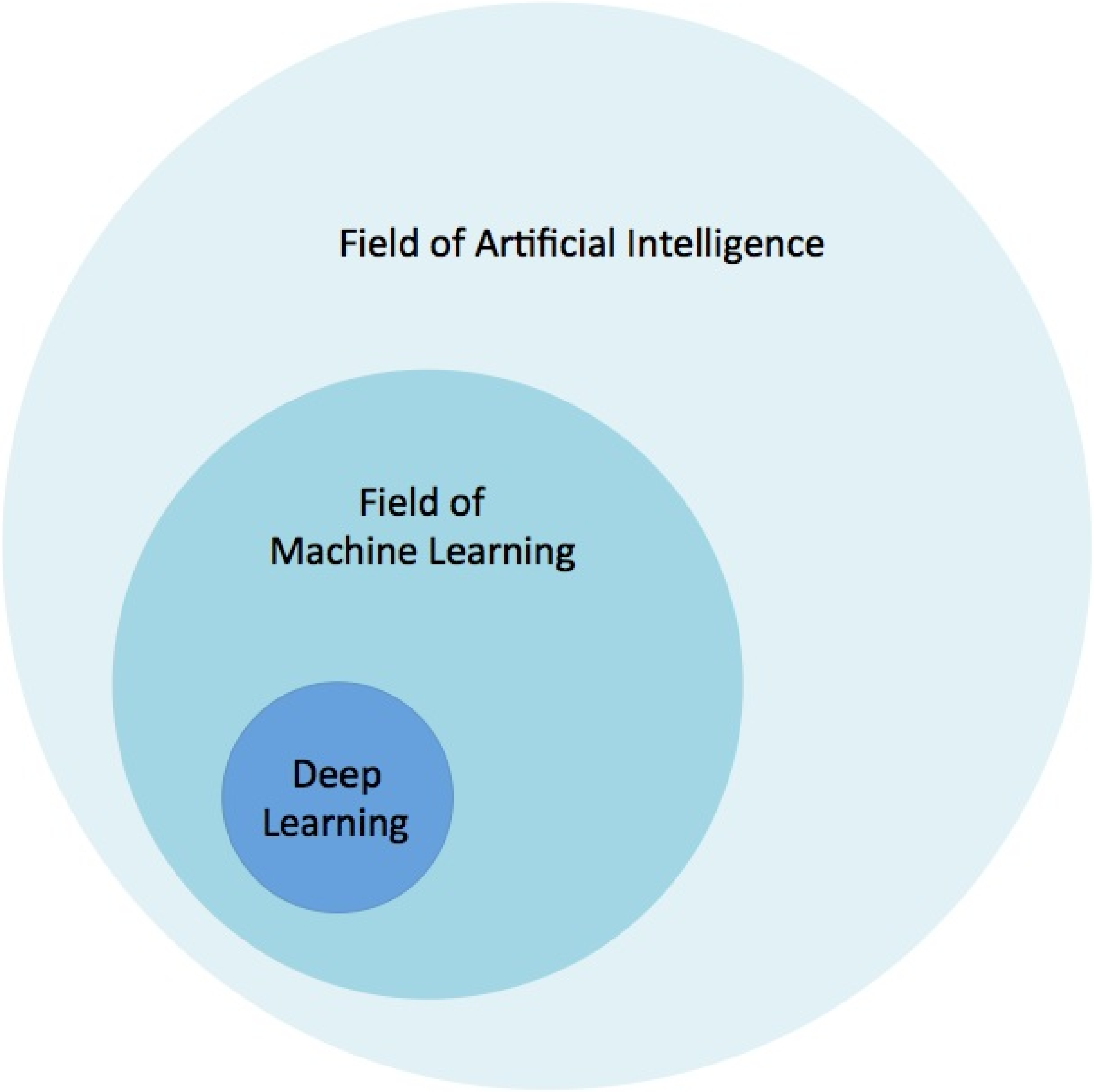
Ask what the net knows... words? Yes Days ?

brackets, quotes, etc

See Lecture 8 3 Hinton

RNN's require much less training data than others on NLP stuff.

A Diagram


Field of Artificial Intelligence
Field of
Machine Learning
Deep
Learning

Machine Learning Compared to Data Science/Mining

Recurrent Neural Networks

Table of Contents

This is Josh's stuff need to augment and rework and generalize

Recurrent neural networks

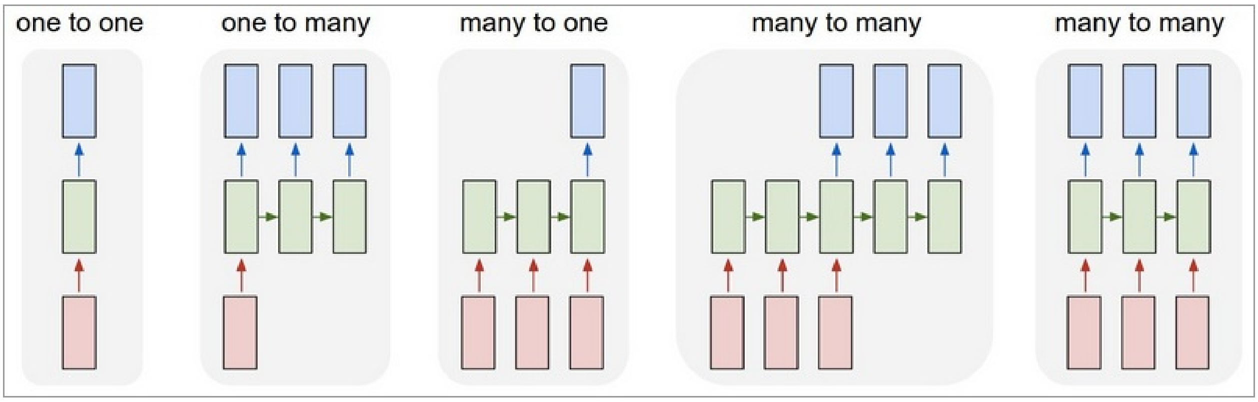
Recurrent neural networks

- 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

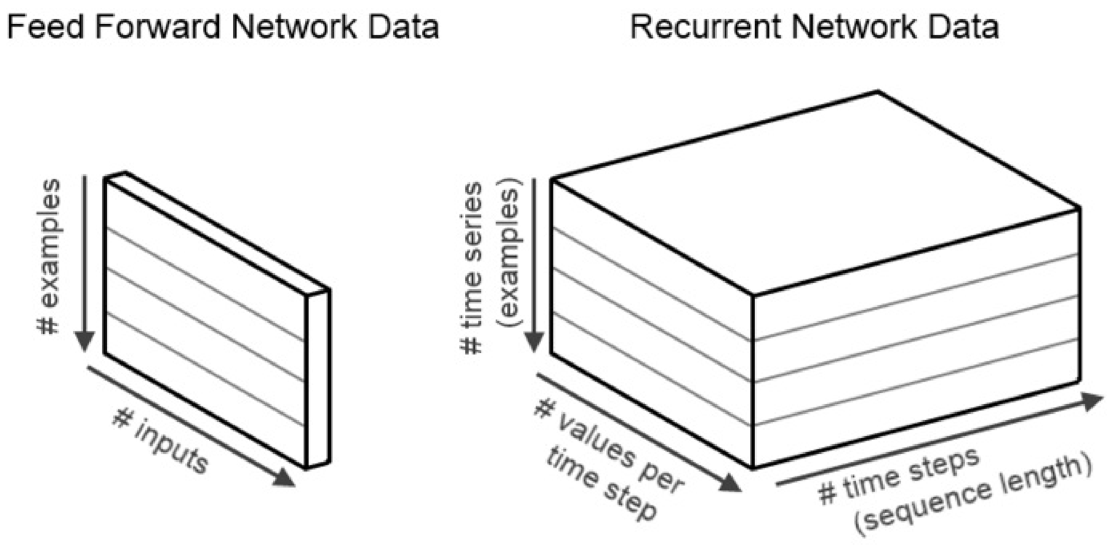
Example: PhysioNet Raw Data

- 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

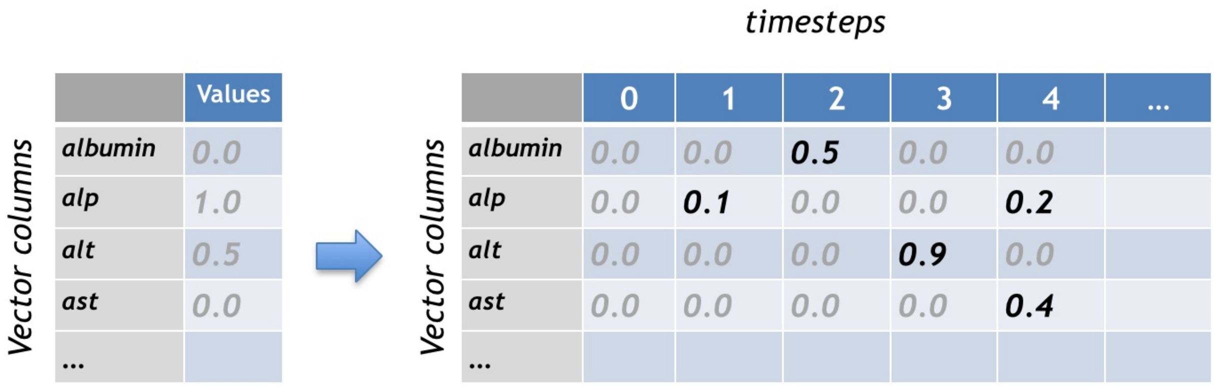


- Input was 3D Tensor (3d Matrix)
  - Mini-batch as first dimension
  - Feature Columns as second dimension
  - Timesteps as third dimension
- PhysioNet: Mini-batch size of 20, 43 columns, and 202 Timesteps
  - We have 173,720 values per Tensor input

TH- explain batch and minibatch in terms of training

Input Sequence

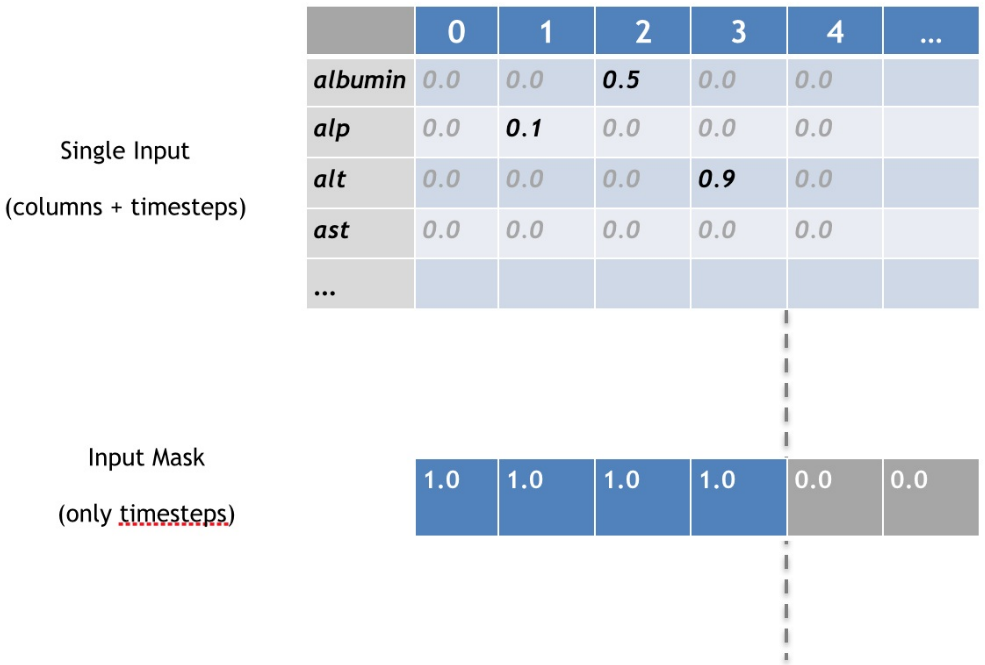
- A single training example gets the added dimension of timesteps per column


timesteps
Vector columns
Vector columns

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

	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**	
...						

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

ETL and Vectorization

Table of Contents

-- this chapter sets up datavec needed for UCI sequence data lab

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

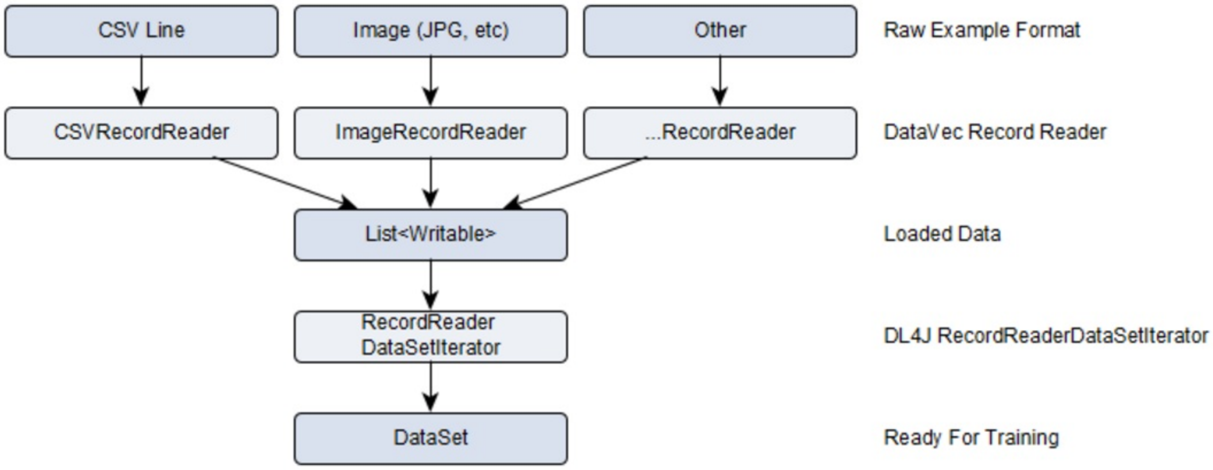
Major Facets of DataVec

- 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

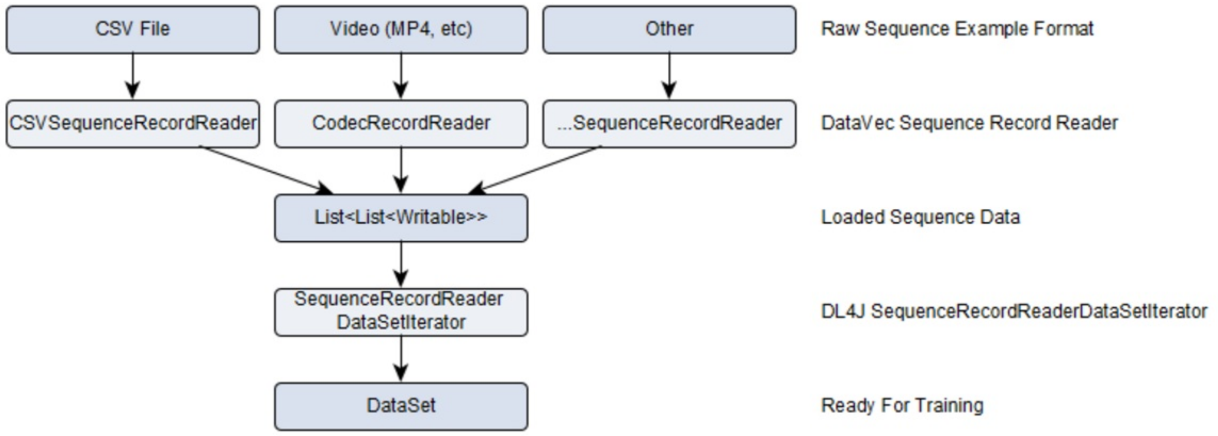
DataVec Processing Pipelines



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

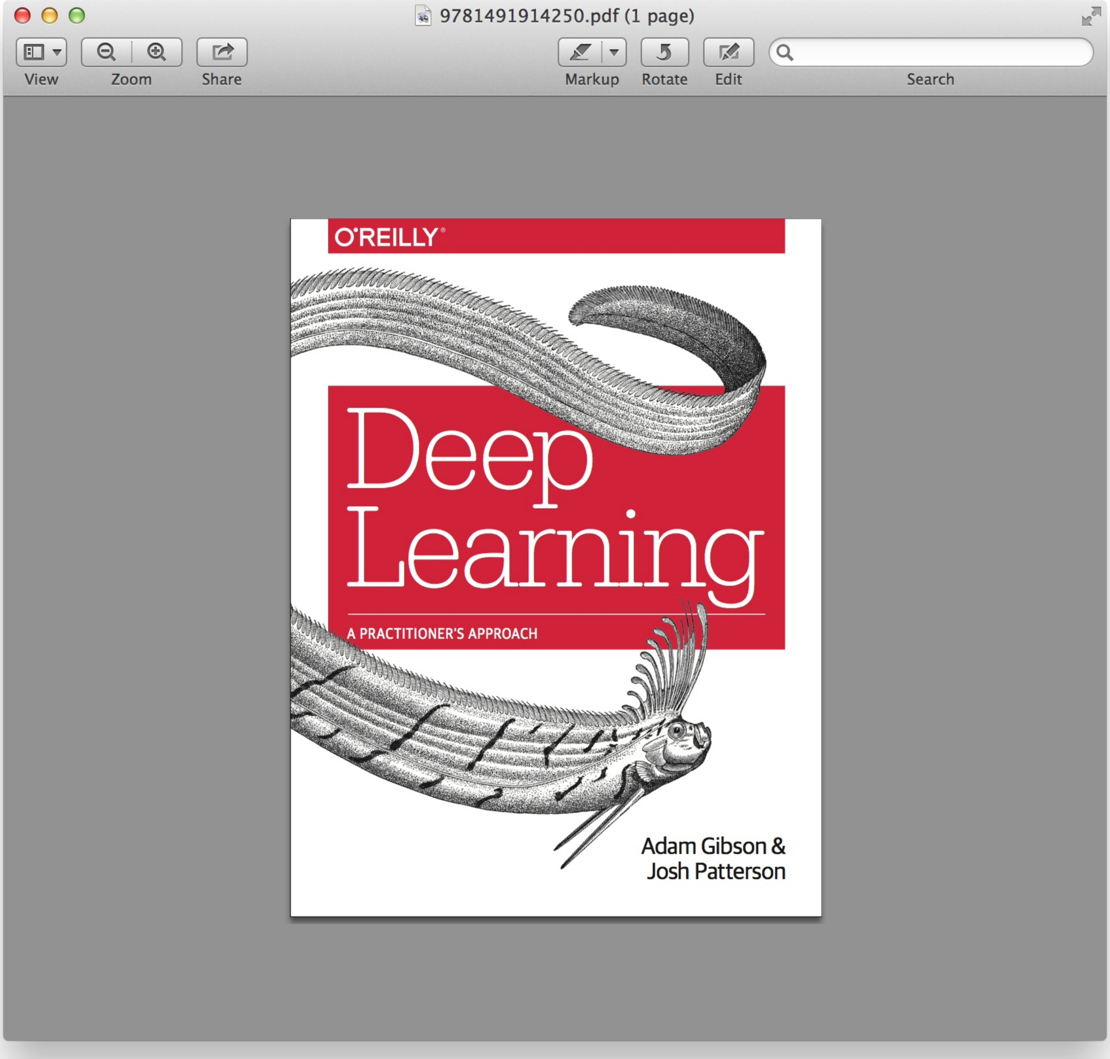
- When we “standardize” a vector we subtract a measure of location (minimum, maximum, median, etc)
    - and then divide by a measure of scale (variance, standard deviation, range, etc).
    - Zero Mean, Unit Variance

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?


9781491914250.pdf (1 page)
View
Zoom
Share
Markup
Rotate
Edit
Search
O'REILLY
Deep Learning
A PRACTITIONER'S APPROACH
Adam Gibson & Josh Patterson

Data Ingestion Case Study: Text

Table of Contents

Basic Concepts of Text Representation for Neural Networks

- 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

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 continued

- 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();
  ```

```



-------------------
<div style="page-break-after: always;"></div>

# Bag of Words Example Continued
```

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

```



-------------------
<div style="page-break-after: always;"></div>

# Bag of Words Example Continued

* 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"));
```

```
-------------------
<div style="page-break-after: always;"></div>

# NGrams
* Contiguous sequence of n items from a sequence of text

Example "It is the year 2016"

Bi-grams "It is" "is the" "the year" "year 2016"
Tri-grams "It is the" "is the year" "the year 2016"

-------------------
<div style="page-break-after: always;"></div>

# NGram uses

* Provide more context than Bag of Words
* Used in some Neural Net for Speech Recognition to narrow the scope of prediction
  * RNN predicts next word out of top x percent of trigram for previous 2 word pre
dictions


-------------------
<div style="page-break-after: always;"></div>

# NGram code Example
```

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 tokens = factory.create(toTokenize).getTokens(); log.info(tokens.toString());

```
Output
```

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

```
```

Word2Vec

- Model for word embeddings
- Vector Space
- Each word in Corpus => Vector in Vector Space
- Relative location of word in vector space denotes relationship
  - Boy->Man Girl->Woman

Word2Vec

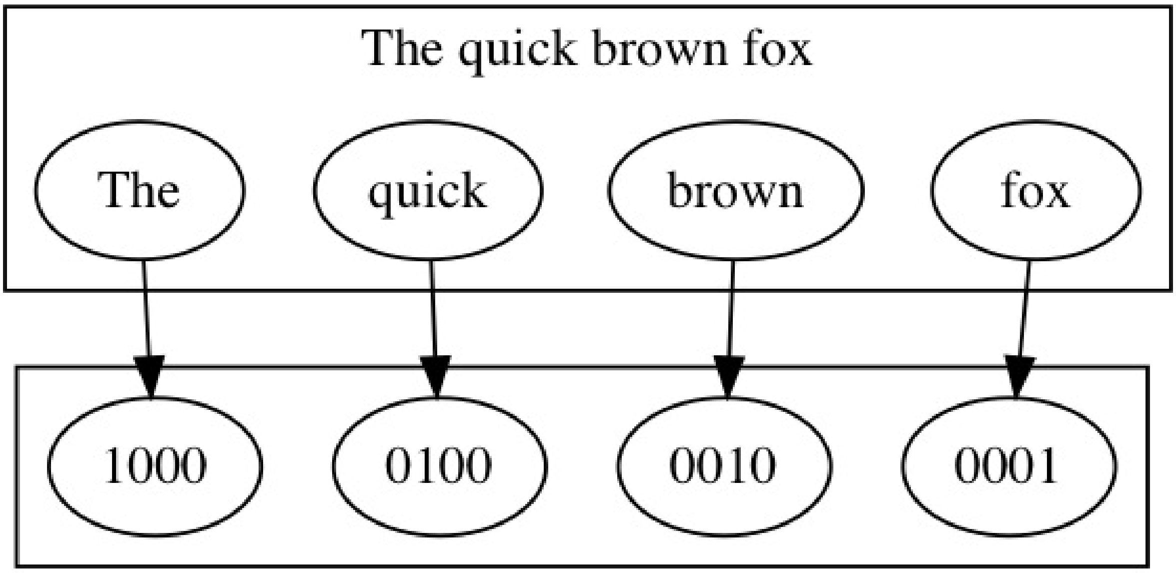
- Model for word embeddings
- Vector Space
- Each word in Corpus => Vector in Vector Space

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 probility of each word occuring within say 5 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
  - King - Man + Woman = (?) Queen

One-hot encoding

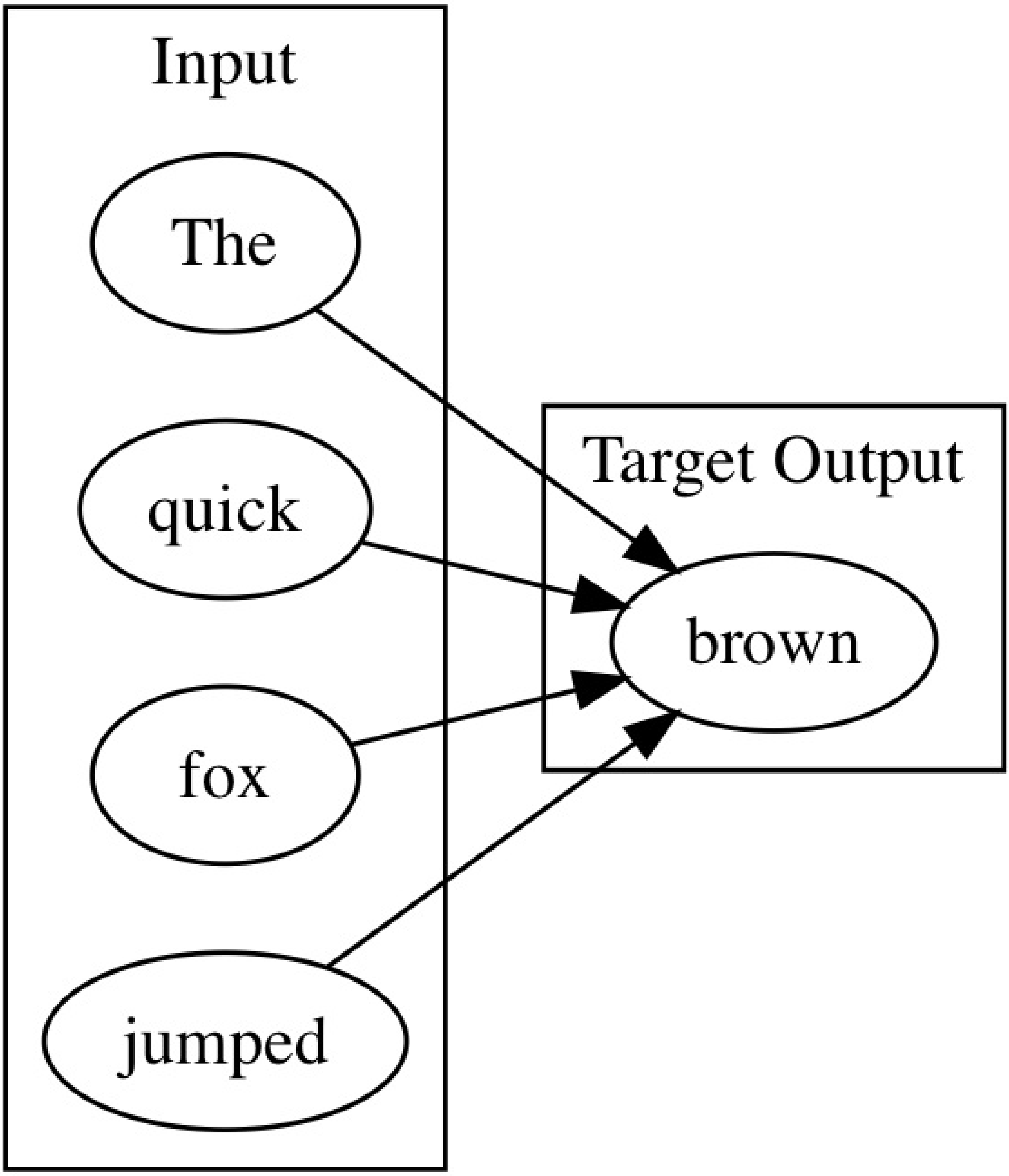
- Vector, the size of the vocabulary, all 0's except for 1



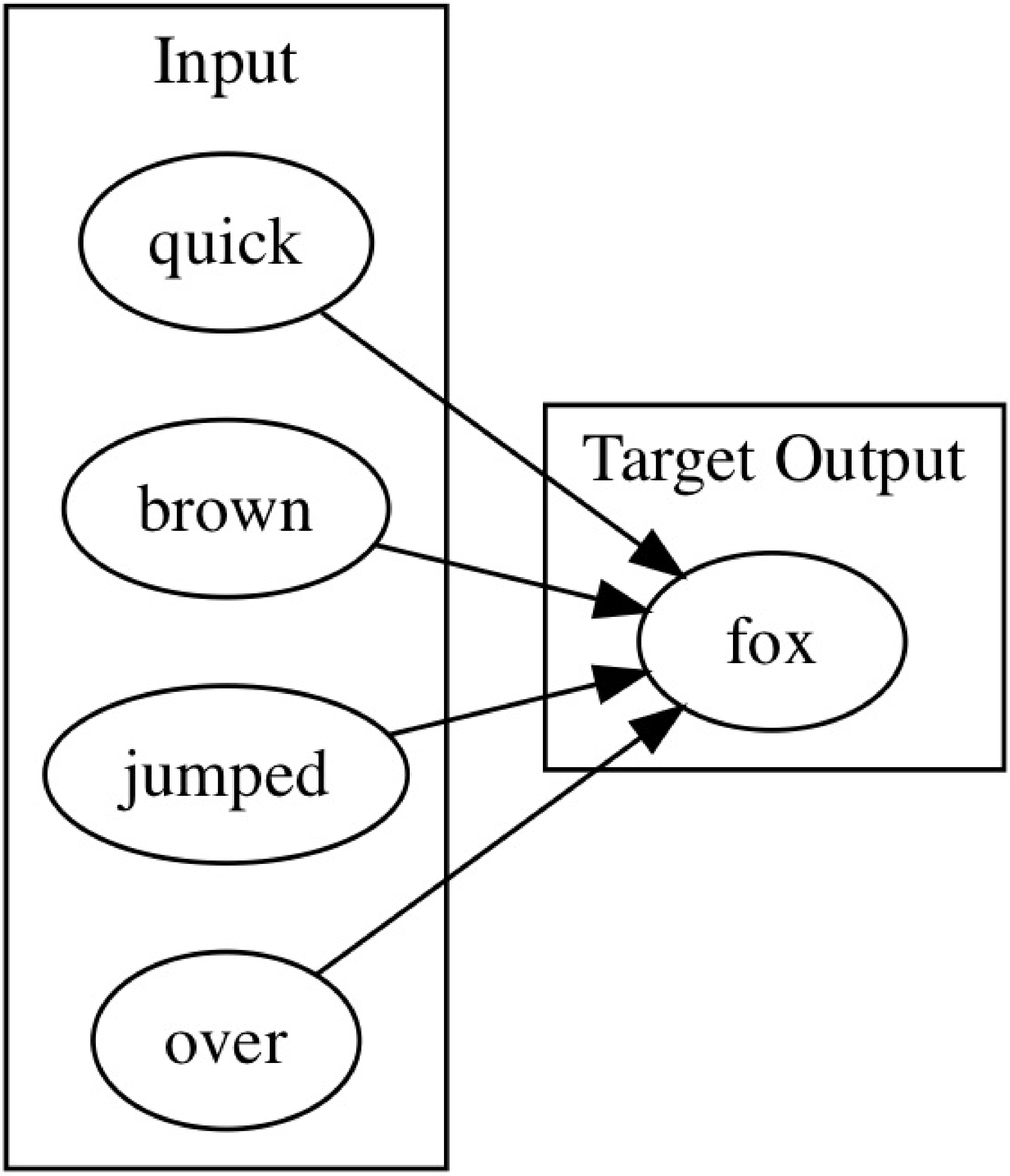
Two Methods for building word2vec

- CBOW
  - w1,w2,w4,w5 as input to Neural Net
    - Context words
  - Train net with w3 as target
    - Focus word
- SKIP GRAM
  - Reverse of CBOW
  - Input is Focus word
  - Output is Context Words

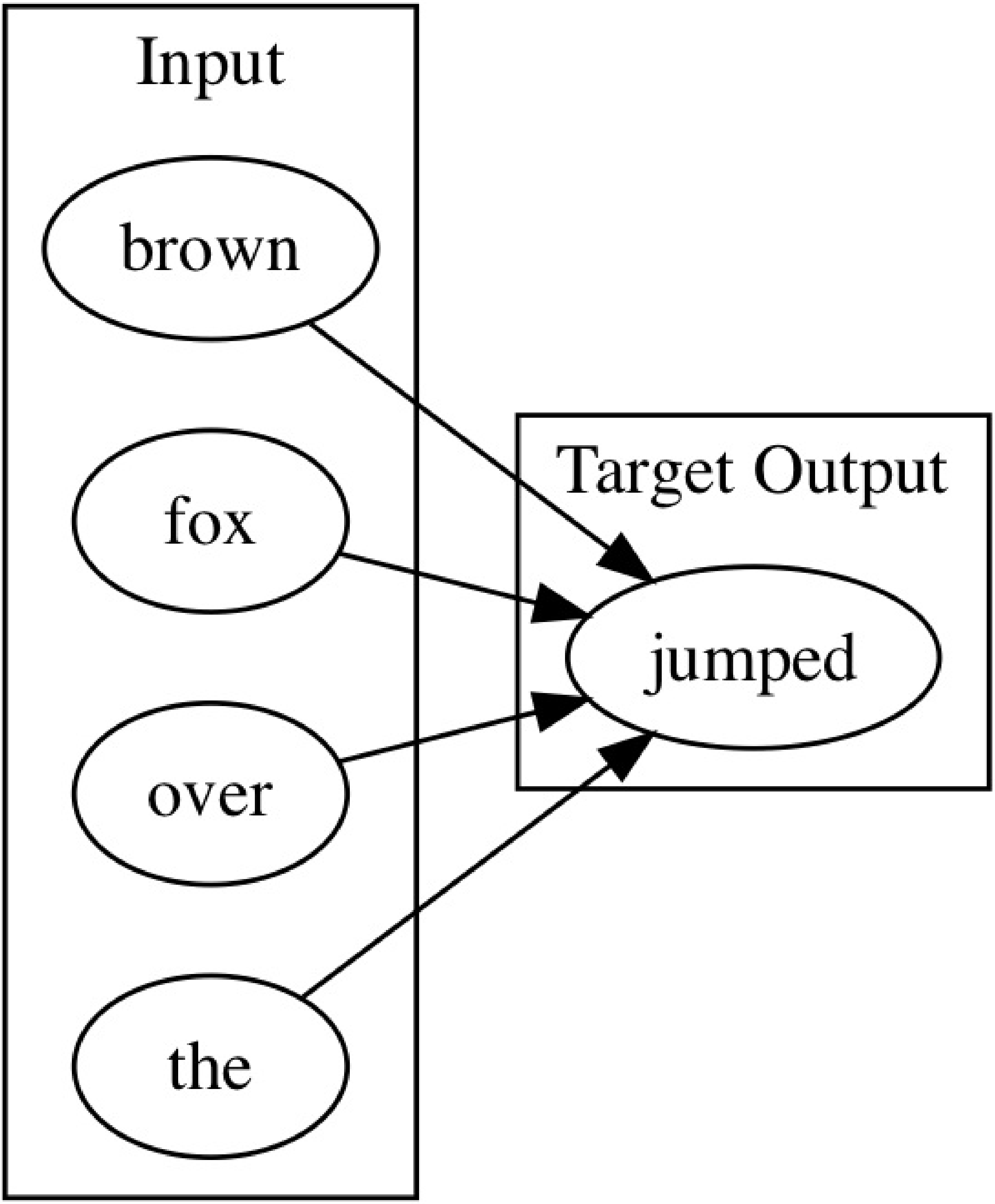
CBOW visually



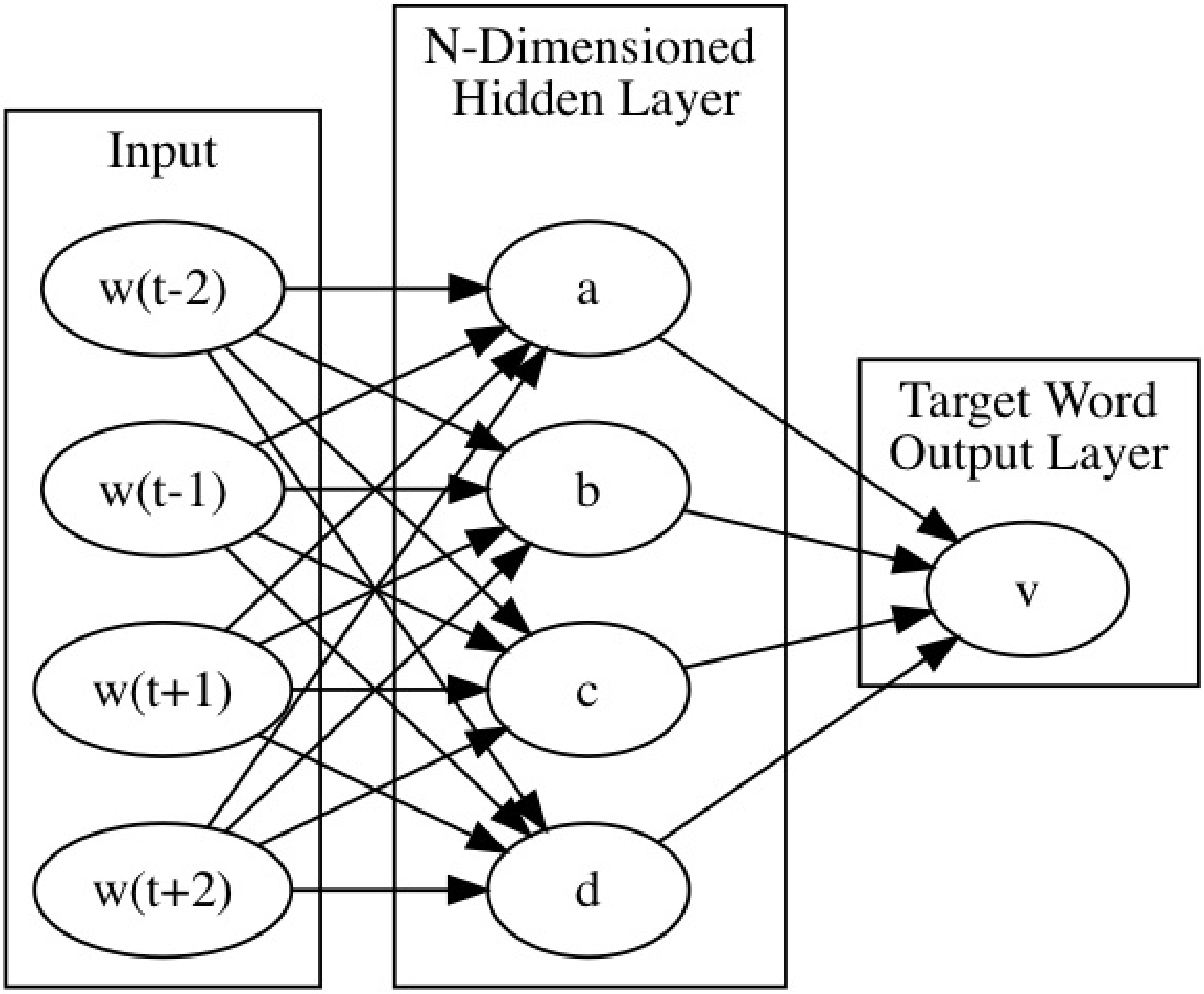
CBOW visually



CBOW visually



CBOW visually



GloVE

- Vector Representation
- word-word co-occurence algorithm for generating vector

Text as Sequence of Characters

Text can be treated as sequence of characters, and neural network can be trained to answer the question. Given input character X predict the next character, and repeat.

Why choose character as unit of analysis vs word?

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

SubTree in tree of all character strings

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

In an RNN each node is a hidden state vector.

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.

Using Recurrent Neural Networks to write weather forecast

In this Lab you will write a RNN to train on Weather Forecasts. As it trains it will output a weather forecast. Watch as it learns words as sequences of characters along with sentence structure and more.

Refer to your Lab Manual.

```
#
```

```
#
```

GloVe