A Quick Introduction to Machine Learning

Paul Rodriguez
(and Mai Nguyen)

SDSC
Overview

• Terminology and Key concepts
• Modeling and Machine Learning
• Main Activities of Modeling
• R and HPC
• Deep Learning and Digit Recognition
Lots of Terms:
Lots of Terms: what are the key ideas?
Lots of Terms: what are the key ideas?

Artificial intelligence
Lots of Terms: what are the key ideas?

Some human-like processing
e.g. face detection, person tracking, object recognition
Lots of Terms: what are the key ideas?
Lots of Terms: what are the key ideas?

gathering data, looking for patterns and associations –
e.g. who buys what products?
Lots of Terms: what are the key ideas?

BIG DATA:

* e.g. social networks, internet activity, big science, etc...
Lots of Terms: what are the key ideas?

BIG DATA:
e.g. social networks, internet activity, big science, etc…

PREDICTIVE ANALYTICS:
predicting unobserved data,
e.g. recommending movies
(in contrast to classical inferencing about in-sample statistics)
Lots of Terms: what are the key ideas?
Lots of Terms: what are the key ideas?

improving system performance with data
e.g. statistical learning,
e.g. models with algorithms for fitting parameters
Machine Learning Steps

- Gather & Prepare data
- Explore data – e.g. know your variables
- Build Model – e.g. simple and complex
- Evaluate

*then perhaps iterate*
Gather and Prepare Data

- Cleaning
- Transforming
- Organizing the data matrix
  (aka ‘data wrangling’ or ‘data munging’)
- Variable Selection/Dimension Reduction
Explore Data – know your variables

• Summary statistics
• Check missing values
• Visualize:
  Plot pairwise correlations
  Plot histograms
Summary Statistics on Variable in R

• Summary statistics on individual variables
  \( (df \text{ is a data frame}) \)

```r
> mean(df$MinTemp)
[1] 7.265574
>
> var(df$MinTemp)
[1] 36.31026
>
> sd(df$MinTemp)
[1] 6.0258
>
> summary(df$MinTemp)
          Min.  1st Qu.   Median      Mean  3rd Qu.     Max.
-5.300   2.850   7.900   7.743   12.800   20.900
```
Histogram in R

```r
hist(df$Sunshine, col="lightblue", main="Histogram of Daily Sunshine", xlab="Sunshine (Hours)")
```
Pairwise Correlation Plot in R

```r
ggpairs(df[c("Sunshine","Pressure9am","Humidity9am","Temp9am")])
```
Machine Learning Models

- Classification
- Regression/Predictive
- Cluster
- Matrix Factorization

- Bayesian (i.e. learning probability distributions)

Supervised (dependent variable or outcome labels given)

Unsupervised (no labels)

Statistical but comes up in HPC settings
A data example: Home Runs and W-L percent

Number of Team HRs 2012 (call it ‘x’)

W-L %
(call it ‘y’)
Recall Linear Regression is Fitting a Line

**the Model:** \( y = f(x, b) = bo * 1 + b_1 * x \)
Recall Linear Regression is Fitting a Line – to minimize error

**the Model:** \( y = f(x, b) = b_0 + b_1 x \)
Can we just classify winners vs losers based on home runs?

W-L %
(call it ‘y’)

Number of Team HRs 2012 (call it ‘x’)
Classification uses labelled outcomes

Winners

Losers

Number of Team HRs 2012 (call it 'x')
Classification uses labelled outcomes

With one input dimension make a cut off – eg HR > 150
A Linear Model for Classification

- 2 classes: +1=Black (WL%≥.5)  -1=Red (WL%<.5)

Q: Classify winning records based on HRs and ERA?

![Scatter plot showing number of team HRs vs. ERA for 2012.](image)
A Linear Model for Classification

- 2 classes: +1=Black (WL%>=.5)  -1=Red (WL%<.5)

Q: Classify winning records based on HRs and ERA?

![Graph showing classification model with HRs and ERA data points and a threshold line.](image-url)
Model Choices

• What kinds of functions to use
  • e.g. Linear vs NonLinear

• What to Optimize
  • Minimize Prediction Error
  • Minimize Classification Errors
  • Maximize Probabilities

• How to Find Parameters
  • Search space of solutions
  • Constraints and Assumptions
Model Space Map – in a nutshell

Linear Reg: \( y = f(X \cdot b) \)

- **Penalized Reg. (Ridge)**: add constraints on \( b \) sizes
- **Piecewise Linear Reg.**:
  - find splits in variables

- **Logistic Reg.**:
  - squash \( f() \) to 0 to 1 class
  - choose class based on \( P(y|x) \) and assume independence
  - Naïve Bayes
  - add layers and back-propagate errors
  - Neural Network

- **Perceptron**:
  - don’t assume indep., get \( P(y,x) \)
  - Bayes Network
  - transform inputs first instead of a ‘hidden’ layer

- **Support Vector Machine**:
  - find splits recursively
  - minimize class errors
  - maximize error margins

- **Classification/RegressionTree**:
  - take ensemble
  - Random Forest
Model Complexity vs Overfitting

more parameters =>
  more complex =>
  more potential to overfit

(so use training and test datasets)
Modeling Recommendations

• Start simple
• Consider trade off as you go more complex
• Find what works in your domain
• Find what works for this model

• R, Python, Matlab: scripting languages with train/predict/test functions
• Weka, KNIME: GUI tools
Pause
Clustering Idea

- Given a set of data can we find a natural grouping?

Essential R commands:

```R
D = rnorm(12, 0, 1) # generate 12 random normal
X1 = matrix(D, 6, 2) # put into 6x2 matrix
X1[,1] = X1[,1] + 4; # shift center
X1[,2] = X1[,2] + 2;
```

#repeat for another set of points

# bind data points and plot
plot(rbind(X1, X2),
     xlim=c(-10,10), ylim=c(-10,10));
Clustering

• **A good grouping implies some structure**
  Interpret and label clusters
  Characterize new points by the closest cluster

• **Kmeans is a standard algorithm**
  Get distances of all points to K cluster centers
  Assign points to closest center
  Recalculate cluster centers
Imagine these 2 dimensional input spaces: Which of these is easy or hard to cluster? (no class labels)
Potential clusters

A

X1

X2

easy

B

X1

X2

easy

C

Hard?

X1

X2

D

easy, 4 clusters

X1

X2
Now imaging there are two classes

A

B

C

D
Potential clusters

A

X1

X2
easy

B

easy, but not match classes

C

hard, and not match classes

D

easy, 4 clusters match 2 classes
Which are easy or hard to classify?
(ie separate red or blue with lines)

A

X1

X2
easy separation

B

X1
easy still

C

X1

X2

harder (nonlinear)

D

X1

X2

harder (nonlinear)
Which are easy or hard to classify?
(ie separate red or blue with lines)

A

X1

X2
easy separation

B

X1

X2
easy still

C

X1

X2
harder (nonlinear)

D

X1

X2
harder (nonlinear)
Which are easy or hard to classify? (ie separate red or blue with lines)

A

B

easy separation

easy still

C

In summary: No easy relationship between clusters and classification

D

harder (nonlinear)
Pause
Matrix Factorization:

*Given a numeric matrix, can we reduce the number of columns?*
Matrix Factorization:

Given a numeric matrix, can we reduce the number of columns?

- Yes, if features are constant or redundant
Matrix Factorization:

Given a numeric matrix, can we reduce the number of columns?

• Yes, if features are constant or redundant
• Yes, if features only contribute noise (conversely, want features that contribute to variations of the data)
Example: Athletes’ Height by Weight

Find a line that aligns with the data.
Example: Athletes’ Height by Weight

Find a line that aligns with the data.
Find a line that aligns with the data.

- Weight - Kg (mean centered)
- Height - cm (mean centered)

*the most variance (or spread)*
Find a line that aligns with the data.

\[ H = 0.8 \times W \]
Find a line that aligns with the data.

\[ H = 0.8 \times W \]

Note that (0,0) and (1,0.8) are points on the line.
The next direction of most variance.
New axis (AKA features) defined as combinations of old features
possibly reduce dimensions

project all points to first axis
it keeps much of the variance
• Best Known Factorization Algorithms:
  SVD (singular value decomposition)
  PCA (principle component analysis)

SVD more generally works on non square matrices
• pause
To get a Compute Node:

[Unix]$ : srun --partition=debug --pty --nodes=1 --ntasks-per-node=24 -t 00:30:00 --wait=0 --export=ALL -A your-account /bin/bash

At Unix prompt at a Compute Node (not a login node):

[Unix]$ module load R
[Unix]$ R (this gets an interactive R session)

quit() to exit R

exit to exit the compute node
Running Jupyter notebook on Comet

1. Login to comet
2. Access compute node: `srun --partition=debug --pty --nodes=1 --ntasks-per-node=24 -t 00:30:00 --wait=0 --export=ALL -A your-account /bin/bash`
3. Start singularity shell
   1. `module load singularity`
   2. `IMAGE=/oasis/scratch/comet/zonca/temp_project/datascience-notebook-e1677043235c_fixjulia_keras_tf.img`
   3. `singularity exec $IMAGE jupyter notebook --ip=*`
4. on local machine, in browser url edit box, enter the http string shown, but replace localhost with comet-XX-XX.sdsc.edu
5. Open R-introHPC.ipynb or LabMNIST_Final.ipynb
6. After logging out in browser shutdown notebook on Comet with Ctrl-C
Quick-R-hpc-intro

April 4, 2018

1 In this introduction we will explore some useful R functions for data preparation. We will look very quickly at clustering and classification

In [1]: #1. First read the data from a CSV file into an R dataframe

\[
\text{W_df_orig = read.table('weather_orig.csv', headr=TRUE, sep="," , stringsAsFactors = TRUE)}
\]

dim(W_df_orig)

1.366 2.24

2 Quick way to view some rows:

In [2]: head(W_df_orig)

<table>
<thead>
<tr>
<th>Date</th>
<th>Location</th>
<th>MinTemp</th>
<th>MaxTemp</th>
<th>Rainfall</th>
<th>Evaporation</th>
<th>Sunshine</th>
<th>WindGustDir</th>
<th>Wind</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-11-01</td>
<td>Canberra</td>
<td>8.0</td>
<td>24.3</td>
<td>0.0</td>
<td>3.4</td>
<td>6.3</td>
<td>NW</td>
<td>30</td>
</tr>
<tr>
<td>2007-11-02</td>
<td>Canberra</td>
<td>14.0</td>
<td>26.9</td>
<td>3.6</td>
<td>4.4</td>
<td>9.7</td>
<td>ENE</td>
<td>39</td>
</tr>
<tr>
<td>2007-11-03</td>
<td>Canberra</td>
<td>13.7</td>
<td>23.4</td>
<td>3.6</td>
<td>5.8</td>
<td>3.3</td>
<td>NW</td>
<td>85</td>
</tr>
<tr>
<td>2007-11-04</td>
<td>Canberra</td>
<td>13.3</td>
<td>15.8</td>
<td>39.8</td>
<td>7.2</td>
<td>9.1</td>
<td>NW</td>
<td>54</td>
</tr>
<tr>
<td>2007-11-05</td>
<td>Canberra</td>
<td>7.6</td>
<td>16.1</td>
<td>2.8</td>
<td>8.6</td>
<td>10.6</td>
<td>SSE</td>
<td>50</td>
</tr>
<tr>
<td>2007-11-06</td>
<td>Canberra</td>
<td>6.2</td>
<td>16.9</td>
<td>0.0</td>
<td>5.8</td>
<td>8.2</td>
<td>SE</td>
<td>44</td>
</tr>
</tbody>
</table>

In [3]: tail(W_df)

Error in tail(W_df): object 'W_df' not found

Traceback:

1. tail(W_df)

In [4]: str(W_df)  #Quick view of the basic 'structure' of the data frame
head(W_long)
#

# Optional: write.csv(W_cast, file='Weather_castwide.csv')

<table>
<thead>
<tr>
<th>Date</th>
<th>Location</th>
<th>MinTemp</th>
<th>MaxTemp</th>
<th>Rainfall</th>
<th>Evaporation</th>
<th>Sunshine</th>
<th>WindDir90nm</th>
<th>Wind</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-11-01</td>
<td>Canberra</td>
<td>6.0</td>
<td>24.3</td>
<td>0.0</td>
<td>3.4</td>
<td>6.3</td>
<td>NW</td>
<td>NW</td>
</tr>
<tr>
<td>2007-11-02</td>
<td>Canberra</td>
<td>14.0</td>
<td>26.9</td>
<td>3.6</td>
<td>4.1</td>
<td>9.7</td>
<td>E</td>
<td>W</td>
</tr>
<tr>
<td>2007-11-03</td>
<td>Canberra</td>
<td>13.7</td>
<td>23.4</td>
<td>3.6</td>
<td>5.8</td>
<td>3.3</td>
<td>N</td>
<td>NNE</td>
</tr>
<tr>
<td>2007-11-04</td>
<td>Canberra</td>
<td>13.3</td>
<td>15.5</td>
<td>39.8</td>
<td>7.2</td>
<td>9.1</td>
<td>WNW</td>
<td>W</td>
</tr>
<tr>
<td>2007-11-05</td>
<td>Canberra</td>
<td>7.6</td>
<td>16.1</td>
<td>2.8</td>
<td>5.6</td>
<td>10.6</td>
<td>SSE</td>
<td>FSE</td>
</tr>
<tr>
<td>2007-11-06</td>
<td>Canberra</td>
<td>6.2</td>
<td>16.9</td>
<td>0.0</td>
<td>5.8</td>
<td>8.2</td>
<td>SE</td>
<td>E</td>
</tr>
</tbody>
</table>

5  Get factors using SVD

In [8]: # Get numeric columns only
cols_numeric = sapply(W_df, is.numeric) # get column classes as a list
W_dfnum = W_df[, which(cols_numeric)]
dim(W_dfnum)

1.328 2.16

In [9]: # 3 turn it into a matrix
W_matrix = as.matrix(W_dfnum)

In [10]: # 3 mean center data
W_mmcntr = scale(W_dfnum, center = TRUE, scale = FALSE)

In [11]: # 4 run SVD command
Wt = t(W_mmcntr)
list = list(Wt %*% W_mmcntr)

List of 3
$d : num [1:16] 367 307 215 164 109 ...
$s : num [1:16, 1:16] -0.03129 -0.01566 0.03563 0.00638 0.00387 ...
$v : num [1:16, 1:16] -0.0776 -0.2114 0.0446 -0.0778 -0.128 ...

6  Get some kmeans cluster and plot onto first two SVD factors

In [12]: # get kmeans for 4 clusters, with 10 iterations and 1 starting points
k4 = kmeans(W_mmcntr, 4, 10, 1)

# set color scheme
col2use = c('red', 'blue', 'black', 'yellow')
# set cluster assignment in colors
colassignments = col2use[k4$cluster]

W_proj = as.matrix(W_mmcntr) %>% Wt %>% t %>% %>% # project data onto 3 components

3
plot(V.proj[,1], V.proj[,2], col=colassignments, main='data pts project to 1,2 SVD compone

# to plot center points, first project them into components
c3 = kmeanscenters%*% Wavs[,1:3] points(c3[1], c3[2], pch=8, cex=2)

data pts project to 1,2 SVD components, colored by kmeans

In [13]: # Y was created above, use it to select 3 colors

    # get class assignment in colors
colassignments = col2vec(Y)

    plot(V.proj[,1], V.proj[,2], col=colassignments, main='data pts project to 1,2 SVD compone

4
In [16]: #get model predictions (more generally we would use a test set to get prediction accuracy
    Y_pred <- model$result$fitted.values

    #get the indices of predictions NO vs YES
    Y_pred_indices <- which(Y_pred==1.6)
    Y_pred2_indices <- which(Y_pred==1.8)

    #set up NO,YES predictions
    Y_pred_class <- matrix(1, length(Y), 1)
    Y_pred_class[Y_pred2_indices] <- 2

    #show a confusion matrix
    table(Y, Y_pred_class)

    Y_pred_class
    Y  1 2
    1 259 9
    2 31 29

In [18]: #now color the predictions onto the 2 SVD dimensions

    colassignments <- col2lab(Y_pred_class)

    plot(U.proj[,1], U.proj[,2], col=colassignments, main="plot vs project to 1,2 SVD components"
data pts project to 1,2 SVD components, colored by PREDICTED class
• pause
Quick R in HPC
Paul Rodriguez
SDSC
R strengths for HPC

• Sampling/bootstrap methods
• Data Gathering and Preparation
• Particular Statistical procedures that you won’t find implemented anywhere else, e.g.
  Multiple Imputation methods,
  Instrument Variable (2 stage) Regression
  Matching subjects for pairwise analysis
  MCMC routines
R Scaling In a nutshell

• R takes advantage of math libraries for vector operations

• R packages provide multicore, multinode (snow), or map/reduce (RHadoop) options

• However, model implementations not necessarily built to use parallel backends
  • Some models more amenable to parallel versions
Solving Linear Systems: Performance with R, 1 compute node

R:
- glm(Y~X,family=gaussian)  # gaussian regression (like lm)
- glm(Y~X,family=binomial)  # logistic regression (Y=0 or 1)

Data Matrix Size (i.e. square, rowsXcol)

- 1K
- 2K
- 4K
- 8K

Wall Time (secs)

30min

GLM: logistic
Gradient descent

Matrix inverse solutions
GLM: Gaussian
lm()
inverse

QR matrix decomposition solutions
Solve(a,b)
qr()
Machine learning models: Performance on 1 compute node

- GLM: logistic - 20 secs
- SVM classification (support vector machine optimization)
- Kmeans
- Im()
- svd factoring

Data Matrix Size (i.e. square, rowsXcol)

Wall Time (secs)
R multicore

• Run loop iterations on separate cores

install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

allocate workers
R multicore

- Run loop iterations on separate cores

```r
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

my_data_frame = ..... 

my_results = foreach(i=1:24,.combine=rbind) %dopar% 
{  
  your code here 

  return( a variable or object )
}

allocate workers
%dopar% puts loops across cores, (loops are independent)
%do% runs it serially
```
R multicore

• Run loop iterations on separate cores

```r
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

my_data_frame = .....  # Allocate workers

my_results = foreach(i=1:24,.combine=rbind) %dopar% {
  ...  # Your code here

  return( a variable or object )
}

%dopar% puts loops across cores, (loops are independent)
%do% runs it serially  # Specify to combine results into array with row bind

returned items ‘combined’ into list by default
```

SDSC | SAN DIEGO SUPERCOMPUTER CENTER  
UC San Diego
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

my_data_frame = …..

my_results = foreach(i=1:24,.combine=rbind) %dopar% {
  …
  your code here
  
  return( a variable or object )
}

allocate workers

%dopar% puts loops across cores, (loops are independent)
%do% runs it serially

BEWARE: foreach will copy data it thinks is need to every core

returned items ‘combined’ into list by default

specify to combine results into array with row bind
R multinode: parallel backend

- Run loop iterations on separate nodes

```r
install.packages('doSNOW')
library('doSNOW')
...
cl <- makeCluster( mpi.universe.size()-1, type='MPI' )
clusterExport(cl,c('data'))
registerDoSNOW(cl)

results = foreach(i=1:47,.combine=rbind) %dopar% 
{ ... your code here
  return( a variable or object )
}
stopCluster(cl)
```

allocate cluster as parallel backend

%dopar% puts loops across cores and nodes
Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:
   mpirun -N processors My-perl-script

   CPU Core 1
   |
   v
   My-perl-script: get cpu-id & pass it to R

   CPU Core 2
   |
   v
   My-perl-script: get cpu-id & pass it to R

   ...

   R script: process dataset 1
   v
   My-perl-script: get cpu-id & pass it to R

   R script: process dataset 2
   v
   My-perl-script: get cpu-id & pass it to R

   ...

   CPU Core N
   |
   v
   My-perl-script: get cpu-id & pass it to R

   R script: process dataset N

   v
   
   Final R script: combine N outputs
mpirun the 'bundler' perl script on 24 nodes

Normal batch job info

```bash
#!/bin/bash
#---------------------
# slurm script for a batch job on comet
# to run a task on individual cores
#---------------------
#SBATCH --job-name="serial-pack"
#SBATCH --output="serial-pack.%j.%N.out"
#SBATCH --partition=shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --export=ALL
#SBATCH -t 00:30:00

bash

#Generate a hostfile from the slurm node list
export SLURM_NODEFILE=`generate_pbs_nodefile`

#Run job from working directory or do something like this:
#cd /oasis/scratch/comet/$USER/mydirectory/

module load R

#mpirun executes the bundler perl script on each core, the bundler will launch
# R with arguments to indicate which file to process
# NOTE the argument to bundler is the number of R tasks to execute
# it should be = ntask-per-node X nodes  (but >= would work)
mpirun_rsh -hostfile $SLURM_NODEFILE -np 24 ./bundler.pl 48
[train100@comet-ln3 Rpacking_serial]$```
Depending on the cpu-id, execute R and pass an input file

The argument ‘was 48 tasks to do

Get current cpu id and number of processes

my $ntasks = $ARGV[0];
my ($myid, $numprocs) = split(/\s+/, `./getid`); #getid will return cpu id and total processor in MPIjob

for (my $i=0; $i<$ntasks; $i++) {
    if($myid == $i % $numprocs) {
        print "Perl Bundler: $myid -th Task\n";
        my $filei_2use=$i+1;
        system("/opt/R/bin/Rscript R_LinModel_randXY.R $filei_2use");
    }
}
### Example: scaling MCMC

**Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models, Frederico Bumbaca, UCIrvine, et al in print**

<table>
<thead>
<tr>
<th># Individuals</th>
<th>Cores</th>
<th>Individ per Core</th>
<th>Total Minutes (I/O time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 million</td>
<td>30</td>
<td>~ 33K</td>
<td>66 (2)</td>
</tr>
<tr>
<td>10 million</td>
<td>300</td>
<td>~ 33K</td>
<td>76 (5)</td>
</tr>
<tr>
<td>100 million</td>
<td>1,7282 (max)</td>
<td>~ 58K</td>
<td>206 (38)</td>
</tr>
</tbody>
</table>

Using `rhierMnlRwMixture` function in the R package, bayesm running on SDSC Comet with ‘serial packing’ parallelization

5 observations per individual; 20,000 MCMC iterations, including 4,000 iterations for burn-in, and keeping every 10th draw.
Installing your own R Packages

• **In R:**
  
  ```r
  install.packages('package-name')
  ```
  
  (see [https://cran.r-project.org/](https://cran.r-project.org/) for package lists and reviews)

• **on Comet:**
  
  ```r
  install.packages('ggmap',
    repos='http://cran.us.r-project.org',dependencies=TRUE)
  ```

If compiling is required and you get an error, call support
R-studio

- R studio: An Integrated development environment for R on your local machine – good for development

*now available on XSEDE through Jetstream cloud*

![R-studio interface](image-url)

- Menu tab
- Edit window to Build scripts
- R console
- Environment: Information on variables and command history
- Plots, help docs, package lists
Other R packages:

- Rspark - R interface to Spark
- pdbR - higher level over R-MPI, distributed matrix support and other
  (better for dense matrices vs Spark)
- Rgputools – GPU support
- Ff, bigmemory; Revolution Scale R – map data to files
distribute data across nodes
Spark Core

Hold data **in memory** across nodes

Spark MLlib

D1, D2, Dn

distribute data across nodes
Spark MLlib

Hold data in memory across nodes

Run code on each part and gather as requested
Distributed implementations of common ML algorithms and utilities

APIs for Scala, Java, Python, and R

Scales well for independent processes
• pause
Quick Deep Learning
Paul Rodriguez SDSC
Recall Linear Regression is Fitting a Line

The Model: \( y = f(x, b) = b_0 \cdot 1 + b_1 \cdot x \)

Number of Team HRs 2012 (call it ‘x’)

W-L % (call it ‘y’)

Intercept = \( b_0 \)

Slope = \( b_1 \)

Parameter to be estimated
Classification uses labelled outcomes

Winners

Losers

Number of Team HRs 2012 (call it ‘x’)
Can do better: fit a nonlinear function

The Model: \( y = f(x, b) = \frac{1}{1 + \exp[-(b_0 * 1 + b_1 * x)]} \)

Use threshold to label prediction:
for \( y > 0.5 \), classify \( x \) as Winner
Can do better: fit a nonlinear function

the Model: \( y = f(x, b) = \frac{1}{1 + \exp[-(b_o \cdot 1 + b_1 \cdot x)]} \)

Winners

Lose

Output is ‘squashed’ [0, 1]
And predictions are like probabilities

Use threshold to label prediction:
for \( y > 0.5 \), classify \( x \) as Winner
Logistic Regression as 1 node network

Call \( b_0 \) bias

\( x_1 \)

Call betas weights

\( w_1 \)

Note: logistic function to transform input to output – call it the ‘activation’ function
Logistic Regression as 1 node network

Call $b_0$ bias

Call $w_1$ weights

Note: call betas weights

Logistic function to transform input to output – call it the ‘activation’ function

Note: other activations are possible,

RELU (rectified linear unit)
Next step: More general networks

Add input variables
More general networks

(assume bias present)

\[ x_1 \quad \rightarrow \quad \text{node} \quad \rightarrow \quad x_2 \]

Add input variables  Add logistic transformations …
More general networks

(assume bias)

\[ X_1 \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \] output value

\[ X_2 \]

Combine transformations!

Add input variables  Add logistic transformations …
More general networks

(assume bias)

\[ X_1 \xrightarrow{\text{Combine transformations!}} X_2 \xrightarrow{\text{output value}} \]

Add input variables  Add logistic transformations ...
But parameter fitting is harder too

(assume bias present)

For each data instance:

\[ \text{Error} = \text{Output} - \text{Target} \]
But parameter fitting is harder too

\[(\text{assume bias present})\]

For each data instance:

\[\text{Error} = \text{Output} - \text{Target}\]

The objective is to minimize

\[\text{Error}\] related to output weights

(same as for logistic regression)
But parameter fitting is harder too

(assume bias present)

For each data instance:

\[ \text{Error} = \text{Output} - \text{Target} \]

But, error signals are only known for output layer, what is error for hidden layer?
But parameter fitting is harder too

(assume bias present)

For each data instance:

\[ \text{Error} = \text{Output} - \text{Target} \]

But, error signals are only known for output layer, what is error for hidden layer?

Minimize \( \text{Error} \) related to output weights, that is also related to hidden weights

(Use derivatives to ‘back-propagate’ errors, “stochastic gradient descent”)

Minimize \( \text{Error} \) related to output weights, that is also related to hidden weights

(Use derivatives to ‘back-propagate’ errors, “stochastic gradient descent”)

Why stop at 1 hidden layer?

• More hidden layers => More varied features, or ‘Deep’ Learning
Train with Care

• More hidden layers => More varied features, or ‘Deep’ Learning

Many more parameters, and error signal at final output layer gets drowned out at lower layers—but penalizing weight sizes, varied activation functions, and more data help!
Feature Transformations, Projections, and Convolutions
A Filter

Many X input, but only 3 connections to each hidden node from the ‘local’ input, i.e. a receptive field

(assume $b=0$)

For node 1 let $W = [w_1 \ w_2 \ w_3 ] = [-1 \ 1 \ -1]$

What values of $x_1, x_2, x_3$ will give maximum node 1 output? (assuming $-1 \leq x \leq 1$)
A Filter

Many X input, but only 3 connections to each hidden node from the ‘local’ input, i.e. a receptive field

(assume $b=0$)

For node 1 let $W = [w_1 \ w_2 \ w_3] = [-1 \ 1 \ -1]$

What values of $x_1, x_2, x_3$ will give maximum node 1 output? (assuming $-1 \leq x \leq 1$)

Informally, node 1 has max activation for a ‘spike’, e.g. when $[x_1, x_2, x_3] = [-1 \ +1 \ -1]$
**A Filter**

Many X input, but only 3 connections to each hidden node from the ‘local’ input, i.e. a receptive field

(assume $b=0$)

For node 1 let $W = [w_1 \ w_2 \ w_3] = [-1 \ 1 \ -1]$

For node 2, 3, etc… copy $W$ for node 1

*What is the hidden layer doing?*
A Filter

Many X input, but only 3 connections to each hidden node from the ‘local’ input, i.e. a receptive field

(assume \( b=0 \))

For node 1 let \( W=\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} = [-1 \ 1 \ -1] \)

For node 2,3, etc… copy \( W \) for node 1

\textit{What is the hidden layer doing?}

Informally, looking for a spike everywhere.

This is essentially a convolution operator, where \( W \) is the kernel.
A Filter

Many X input, but only 3 connections to each hidden node from the ‘local' input, i.e. a receptive field

(assume $b=0$)

For node 1 let $W=[w_1 \ w_2 \ w_3 ] = [-1 \ 1 \ -1]$

For node 2,3, etc… copy $W$ for node 1

What is the hidden layer doing?

Informally, looking for a spike everywhere

This is essentially a convolution operator, where $W$ is the kernel.

Note: sharing weights is like sliding $W$ across input
A Filter

Many X input, but only 3 connections to each hidden node from the ‘local’ input, i.e. a receptive field

(assume $b=0$)

For node 1 let $W = [w_1, w_2, w_3] = [-1, 1, -1]$

For node 2, 3, etc… copy $W$ for node 1

What is the hidden layer doing?

Informally, looking for a spike everywhere.

This is essentially a convolution operator, where $W$ is the kernel.

Note: sharing weights is like sliding $W$ across input

Note: if we take max activation across nodes (‘Max Pool’) then it’s like looking for a spike anywhere.
2D Convolution

Now let input be a 2D binary matrix, e.g. a binary image) fully connected to 1 node

What $W$ matrix would ‘activate’ for a upward-toward-left diagonal line?
2D Convolution

Now let input be a 2D binarized 3x3 matrix fully connected to 1 node

What \( W \) matrix would ‘activate’ for an upward-toward-left diagonal line?

How about:

\[
W = \begin{bmatrix}
-1 & -1 & 1 \\
-1 & 1 & -1 \\
1 & -1 & -1 \\
\end{bmatrix}
\]
2D Convolution

For full image, 1 filter is applied to 1 region in 1 color channel at a time, and then slid across regions (or done in parallel with shared weights) and produces 1 new 2D image (hidden) layer

Convolution Layer parameters:

- filter size depends on input:
  - smaller filters for smaller details
  - 2 layers of 3x3 ~ 1 layer of 5x5
- sliding amount
  - smaller better but less efficient
- number of filters
  - depends on task
  - each filter is a new 2D layer

Convolution Network:

- many layers and architecture options
Large Scale Versions

- Large (deep) Convolution Networks are turning out to be feasible with GPUs (some are 100+ layers)
- Need large amounts of data and many heuristics to avoid overfitting and increase efficiency

Classification layers and output
Large Scale Versions

• Zooming in:

The thickness is the number of different convolutions, i.e. different transformations, sometimes called “channels”.

Each convolution layer uses RELU (rectified linear activation units instead of logistic function) and is followed by Max Pooling layer (over 2D regions with sliding).
Large Scale Versions

- Zooming in:

Last convolution layer is laid out as a vector for input into classification layers. Classification uses dense, i.e. fully connected, hidden layers and output layer.
What Learned Convolutions Look Like

What Learned Convolutions Look Like
Summarizing Deep Layers

• Hidden layers transform input into new features:
  • Feature can be highly nonlinear
  • Features as a new space of input data
  • Features as projection onto lower dimensions (compression)
  • Features as filters, which can be used for convolution

• But also:
  • Many algorithm parameters
  • Many weight parameters
  • Many options for stacking layers
Feature Coding vs Discovery

• Edge detection with Support Vector Machine OR Convolution Neural Network?

• With small datasets and obvious features, SVMs can work well

• But building features is hard, and large classification problems can benefit from common features, so CNNs are better to discover features for multiclass outputs
References

• Book: https://mitpress.mit.edu/books/deep-learning
• Documentation: https://keras.io/
• Tutorials I used (borrowed):
  • http://cs231n.github.io/convolutional-networks/
  • https://github.com/julienr/ipynb_playground/blob/master/keras/convmnist/keras_cnn_mnist.ipynb
• pause
Tutorial

- MNIST database of handwritten printed digits
- The ‘hello world’ of Conv. Neural Networks
- Use Keras front end (high level neural functions) to Tensorflow engine (neural math operations)
- Works with GPU or CPUs
for i in range(8,3):
    im = Image.fromarray(Y_train[i,:,:])
    im.save("Xtrain_num"+str(i)+"_cat="+str(Y_train[i])+".jpg")

plt.figure()
plt.imshow(im,'gray')
plt.show()
print('img load done:
print (time.strftime("%M:%D/%S"))
(5060, 28, 28)
<matplotlib.figure.Figure at 0x2ad45d4f2e8>
3x3 first convolution layer filter and activation
9x9 first convolution layer filter and activation