Conclusions

Practical Machine Learning (with R)

UC Berkeley

Spring 2016

3 Questions

- What did you learn that was unexpected and insightful?
- What did you struggle with the most?
- What are you sorry that we didn’t get to cover? / What are you going to learn on your own?

Class Evaluation

- Please do it now.

Agenda

- **Administrativa**
  - Role Call / 3 Questions
  - Class Evaluations
- **Review/Expectations**
  - Readings
    - **Forecasting Principals and Practice**
      - Chapters 1 "Getting Started"
      - Chapter 2 "The Forecaster's Toolbox"
      - Chapter 6 "Time Series Decomposition"
    - **APM**
      - Chapter 7.1, 13.2 "Neural Networks"
      - Chapter 7.3, 13.4 "Support Vector Machines"
  - Previous Lecture Rewiew
- New Topics

Remaining Topics

Today

- Neural Networks
- Deploying Models
- Recommender Patterns
- SVMs

QUESTIONS

NEURAL NETWORKS / ARTIFICIAL NEURAL NETWORKS (ANN)

Analogy to How the Brain Works

Essential Brain Functions

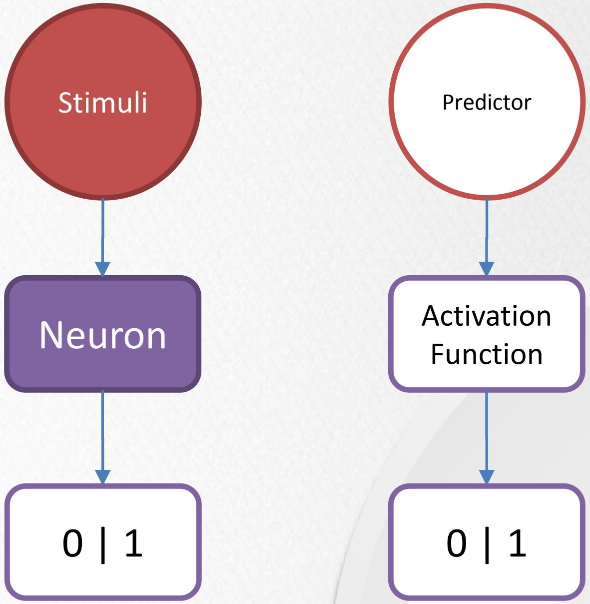
- Neurons receive signals from stimuli
- Neurons either “fire” or not → 0 | 1
- If fired, signal is propagated.

- Neurons can have multiple inputs and outputs
- Neurons can receive and send signals to other neurons.
  - “Network of neurons” result in cognitive function

- Response is continuous valued

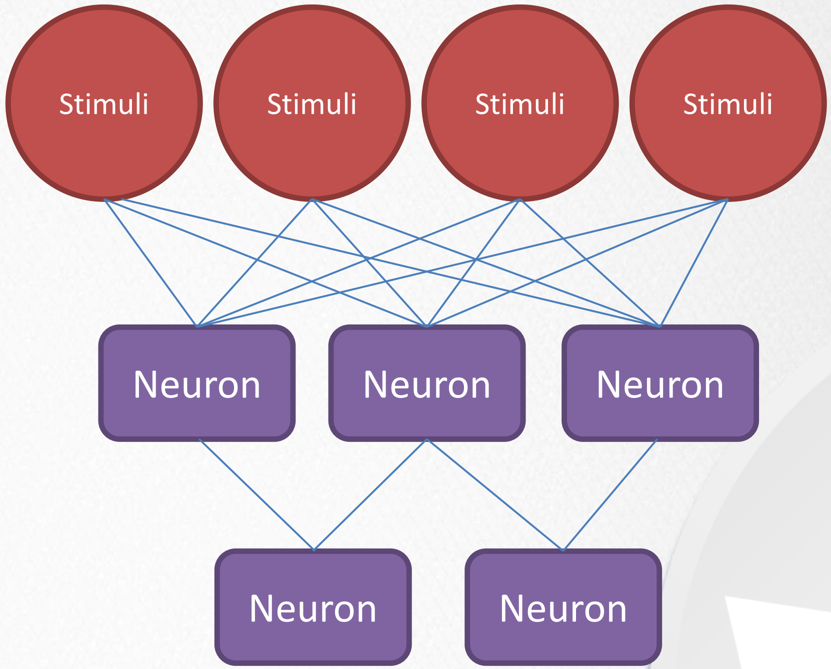
Neuron

- Neurons receive signals from stimuli
- Neurons either "fire" or not → 0 | 1



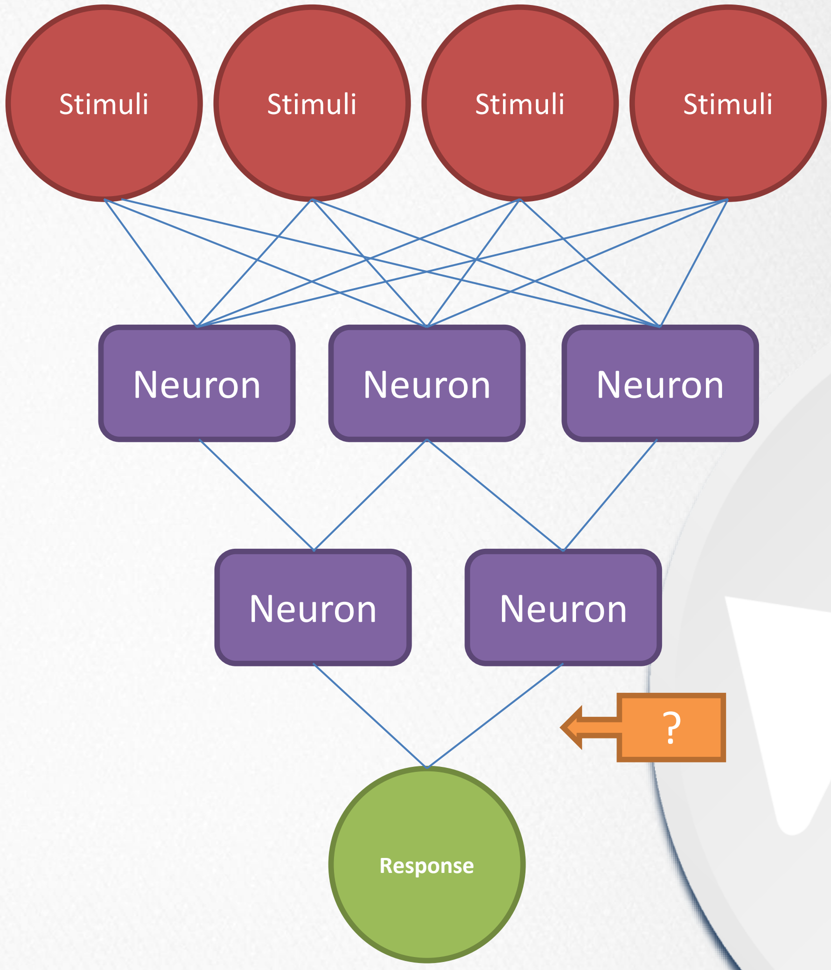
Network

- Neurons can have multiple inputs
- Neurons can send a signal (0,1) to multiple outputs
- Neurons can receive and send signals to other neurons.
  - "Network of neurons" result in cognitive function

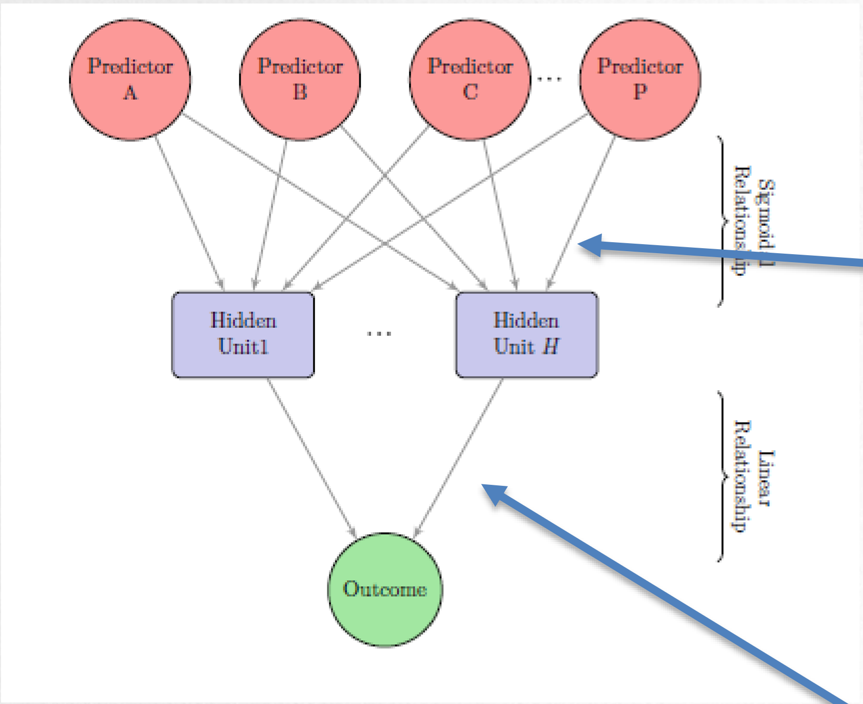


Response

- Response is continuous valued



SINGLE-LAYER FEED-FORWARD NETWORK



Hidden units are comprised of transformation of linear combination of predictors:

$$h(x) = g(\beta_0 + \sum_{i=1}^{P} x_i \beta_i)$$

Where $g(x)$ is a *sigmodial activation* function, resulting in values equal or close to 0 or 1 (often the logistic function).

Outcome is then a regression/classification on the Hidden Units, for example:

$$f(x) = \gamma_0 + \sum_{k=1}^{H} x_k \gamma_k$$

NB: β and γ are called "weights".

Neural Network As a Learning Algorithm

Component	
Loss Function	Various typically: • RMSE (Regression) • Binary 0-1 Loss (Classification)
Restricted Class of Function	• Sigmodial Activation Function (logistic) with • Linear combination of hidden units (linear regression)
Search Methodology	Optimization, typically via Back-propagation

Neural Network Classification

Neural Networks: Variations

- Neural Networks can be made more complex by:
  - Adding hidden units
  - Adding hidden layers
  - Allowing back propagation of neuron signals.

Neural Network: Advantages

- Performs very well
  - yields highly accurate predictions
  - low signal-to-noise
  - Non-linear response
- Easy to add or remove complexity
  - Hidden units
  - Hidden layers
- Modelled after brain function

Neural Networks: Disadvantages

- Not interpretable → difficult to relate inputs to outcomes

- Requires numeric inputs, typically scaled/centered

- All parameters, $\beta's\ and\ \gamma's$ estimated simultaneously
  - A large number of parameters →requires larger data sets.
    $$H \ * (P + 1) + H + 1$$
    - Training times can be long
    - Numerical solution technique:
      - No guarantee of uniqueness
      - Dependent on initial selections of parameters usually small random

- Tends to *overfit*

Solutions To Overfitting

- **Early stopping**: iterative algorithms for solving for the regression equations can be prematurely halted

- **Weight Decay**:
add a penalty for large regression coefficients; large values must have a significant effect on the model errors to be tolerated:

$$\sum_{i=1}^{n} (y_i - f_i(x))^2$$

$$0 < \lambda \leq 0.1$$

ASIDE: WEIGHT DECAY

- This addition of a term to the minimization target is called "shrinkage" or "regularization"
- Tuning parameter … Bias Variance Trade-off
- Can be applied to any technique that directly estimate parameters … linear regression (ridge L1/lasso L2)

Deep Learning

- Buzz word: Rebranding of (artificial) neural networks
  - Training speed improvements (Hinton et. al, 2013)
  - Hardware Improvements
- Requirements
  - Multiple (often) many layers
  - Supervised or unsupervised learning at each layer
  - Each successive layer is a higher level of abstraction

Pixel → Group of pixels → Whisker → Feline Face → House Cat
layer 1 layer 2 layer3

- Higher abstractions obviate feature engineering … to a point.
  - Untransformed input features
  - Large success on unsupervised learning, e.g. can we create a model for cats.

Deep Learning

Benefits	Limitations

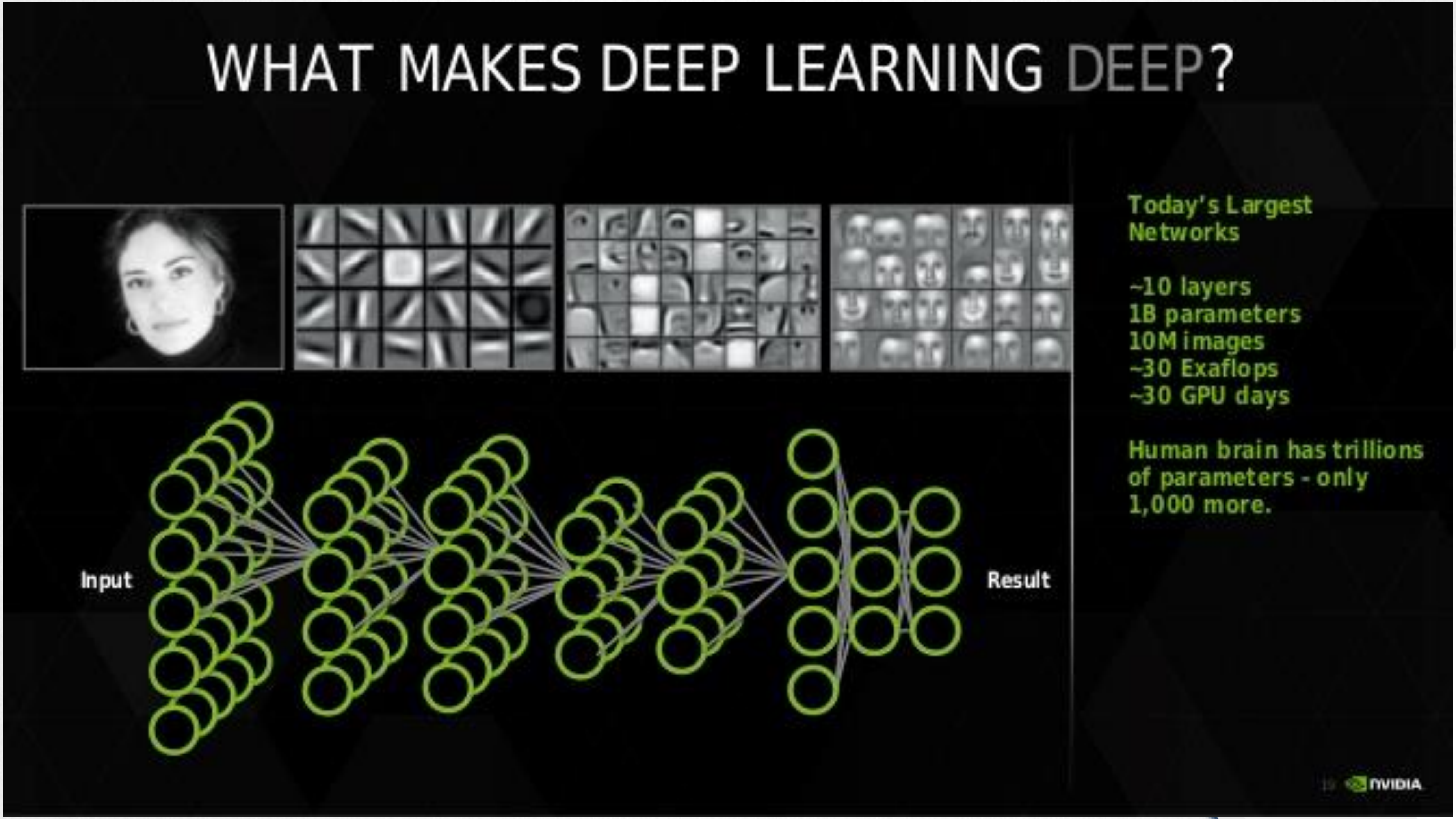
Benefits

- Highly accurate → NN
- Reduced need for manual feature engineering
- Does not necessarily require labelled data
- Exciting area of development ... AI Spring

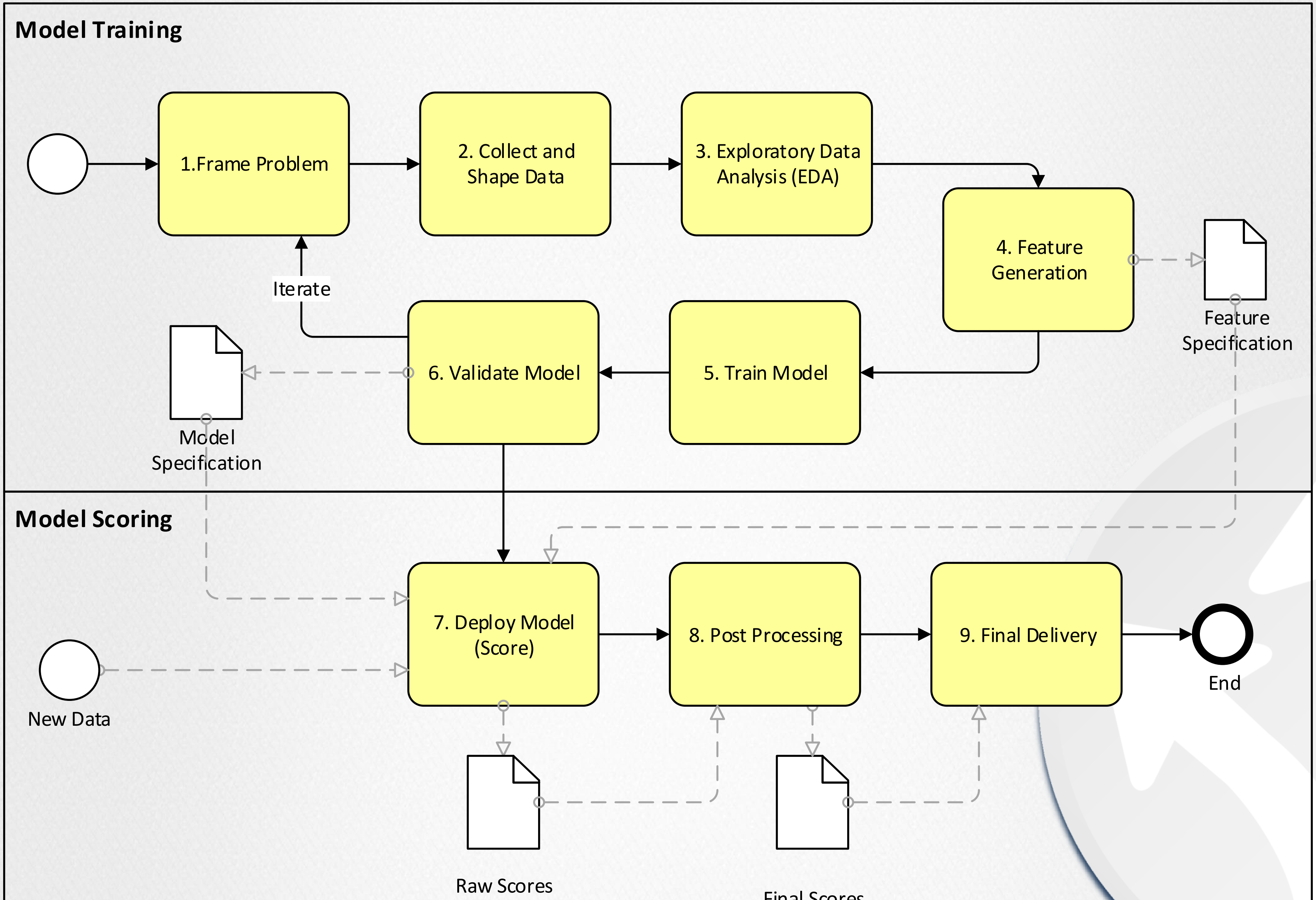
Limitations

- Computationally expensive
  - long training times or
  - need for specialty hardware
- High number of parameters require lots of data
- Really novel?

Deep Learning



DEPLOYMENT



Source: Decision Patterns

Deployment Considerations

- **Access:** How will the model be accessed?
  Online? Excel? In Database? In application?
- **Trigger** : What event when scoring occurs? How often?
- **Inputs**
  - Where data to be scored exists, how obtained
  - How much data?
  - What features can be calculated and cached offline?
- **Outputs**: determine the output of the models
  - **Scores** : raw score, class, class probabilities?
  - **Additional data** : e.g. benefit, prescribed actions
- **Logic**

  Data → [Feature Specifications] → Features →[Model Specifications] →Final Processing
  (online or offline)

Model Access And Usage Patterns

User	Access Point	Deployment Technology	Consideration(s)
Client, Non-technical	Web	Shiny	Interface changes with most model updates
Client, Excel-user	Excel	OpenCPU + VB REST libs	Excel quirks, e.g. Auto calculate
Non-R App. Developer	RESTful API	OpenCPU	Marshalling JSON data to from R
R Developer	Web RESTful API Native R UI R / Rstudio	Shiny OpenCPU Rserve libraries / git / packrat	
Recurrent Process Manual Automated	 command line cron	Rscript + optigrab or	Logging, trapping errors, default parameters, notifications (long processes)
Proprietary Application • Database • BI Application		 SQL Server (SSAS), R Oracle Enterprise	Mileage varies

REVIEW: CREATING AN R PACKAGE

A package is just a directory + conventions for locating code, data, documentation and more

To create a package template

`devtools::create("path/to/mypackage")`

Packages:

- Save models / feature specs objects to `data/` directory:
  - `save`
- Use `roxygen2` for documentation
  - Uses tags: `@export, @parm, @main`, ...
- Use package metadata for
  - Author:, Date:, **Version:**
- Best practice, use `git tag` to match version number

Shiny – What is it

Interactive web framework for analysis and visualization in R

- Application written entirely in R
- ***Reactive*** programming
  - Variable have dependency on other values
  - Update values when dependencies change
- Separated concerns
  - **`ui/ui.R`** (layout, presentation)
    - high-level functions for widgets and layout
  - **`server / server.R`** (application) : application logic
- Well-documented: shiny.rstudio.com
  - Tutorials, examples, gallery …
- Integrated with ***Rstudio*** (`runApp`)
- Uses popular JS libraries for client side reactivity, **DataTable**, **Selectize**, **d3.js.** Search CRAN and Shiny Web sites.

Shiny Resources

- shiny.rstudio.com
- CRAN
- R-bloggers
- Stack-overflow

SHINY SCORER EXAMPLE

openCPU

- Exposes R as a RPC web service:
  - http://myserver/ocpu/path/command/format

- Examples
  - Path:

    ```
    library/stats/
    library/datasets
    ```

  - Command

    R/mean
    data/airquality

  - Format:

    ```
    print | json | csv | tab | md |
    rda | rds | pb | png | pdf | svg
    ```

- **Example: openCPU example**

Shiny vs openCPU

Shiny	openCPU

Shiny

- user-interactive apps
  - single purpose
  - single threaded
- manages state
  - Remembers uses settings between calls
- Easy-to do simple things
- Integration with RStudio

openCPU

- Web service
  - part of larger application
  - analysis deployed to multiple endpoints
  - Batch
- Stateless
  - Inherently multithreaded
- More complex
- Manual configuration

MISCELLANEOUS TOPICS

Formula Tools

- `i` and `poly`

Custom Loss Functions

- Use in-model training → rarely available
- Use case weights → more weight to are
- Use 2nd optimization → another model

RECOMMENDER SYSTEM

Recommender System

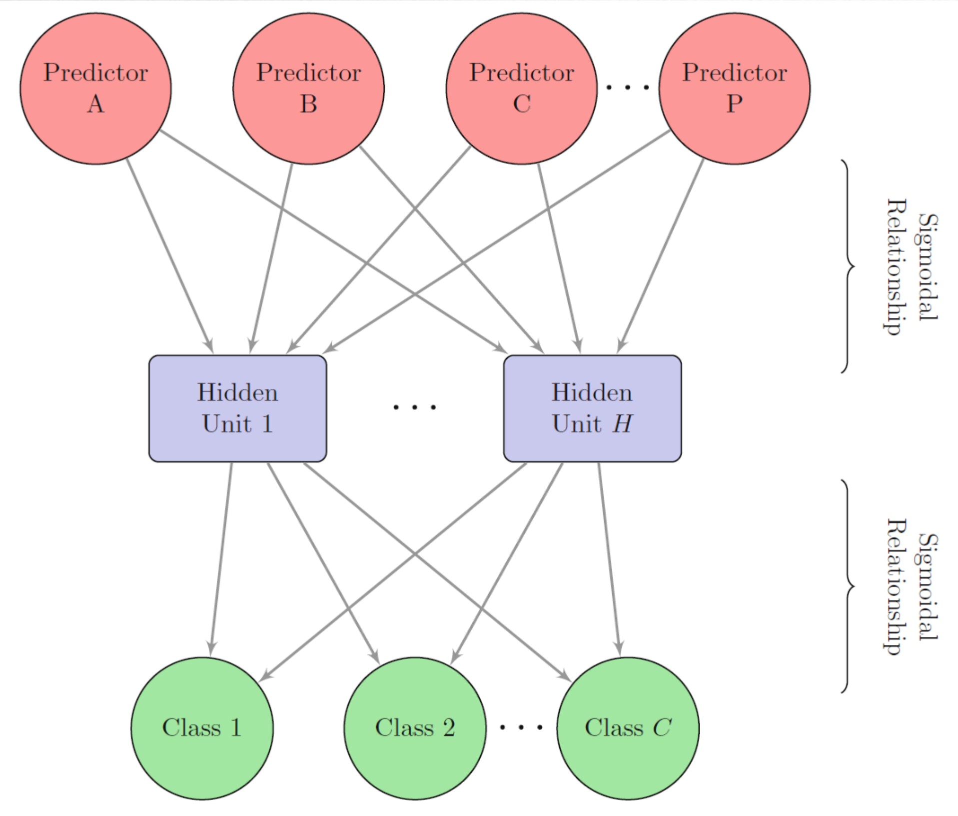
- Goal develop a **Netflix/Amazon** style recommendations system to users

Setup

- Two disjoint sets: **users** and **products**
- Historical interactions:
  - Sales / Rentals
  - Ratings/ Likes /Comments
  - Etc

APPENDIX

Neural Networks: Classification



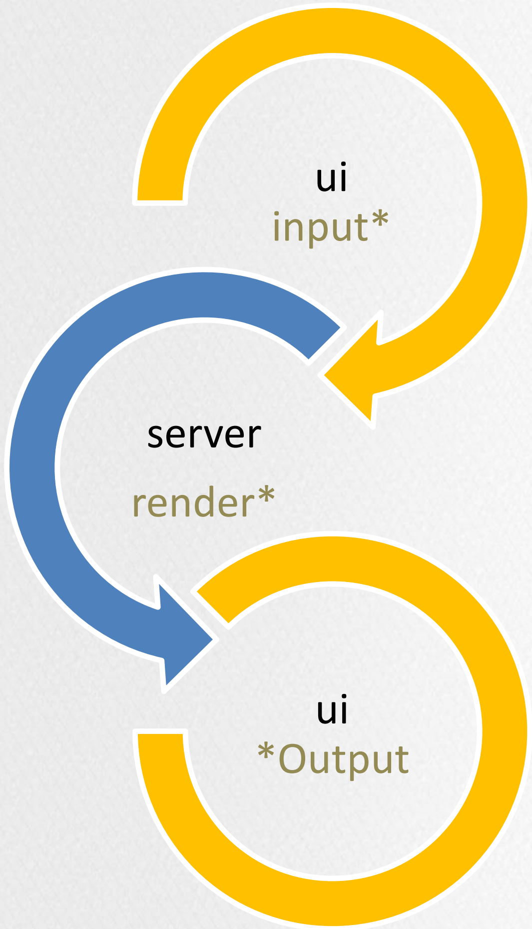
- Add additional responses, one for each class.
- Transform continuous response are transformed by logistic function
- Apply ***Softmax*** function:
  Transform all class probabilities via the soft max function.

$$\hat{p}^*_\ell = \frac{e^{\hat{y}_\ell}}{\sum_{l=1}^{C} e^{\hat{y}_l}}$$

Maps arbitrary real-values

- so that all values sum to 1
- and are between 0 and 1

Shiny Component



ui.R

- HTML Layout:
  - `fluidPage`
  - `navBarPage`
  - `bootstrapPage`
  - `shinydashboard`
- Widgets
  - Input: `input*`
  - Output: `*Output`

Reactivity:
ui redraws *in response to* **user inputs** and **server events**

server.R – logic/data/results

- Manage input and output
  - `input$`*`name`*
  - `output$`*`name`*
- Manages reactivity
  - `observe,observeEvent`
  - `Isolate`
- Creates data/results:
  - `render*`

Reactivity:
server *responds to changes in* **user inputs**

Shiny: Best Practices

- Best way to learn ***shiny*** is to do ***shiny***

- Shiny applications can get complicated quickly
  - Designed for simple, single-page, single-thread apps
  - Goal: Reduce complexity

- All shiny functions are **camelCase**
  - Separate concerns
  - Keep related things close together

- use `source` as needed

- Separate common elements for both `ui.R` and server.R into common file and `source` them or place them in a package

Shiny: Best Practices

- **ui.R**
  - Program model uses **functions** and **names**
  - All ui widgets must have unique name even if identical element is used twice.
  - Naming can be confusing → adopt a naming convention.
  - For complex applications, separate widgets and layout
    - layout → ui.R
    - widgets → widgets.R

- **Server.R**
  - Program model uses named lists
    - All inputs accessed by `input` list, must respond appropriately to empty inputs
    - All rendered objects must be placed in `output` list
  - Dynamic inputs use `renderUI` (`server.R`) & `uiOutput` (`ui.R`)
    - If you use any dynamic inputs, use all dynamic inputs

Shiny: Best Practives

- **Development**
  - Develop within Rstudio

```
library(rstudioapi)
runApp( launch.browser=rstudioapi:::viewer ) # or
options("shiny.launch.browser" = rstudioapi:::viewer )
```

  - `runApp` can reload with ui.R errors / must be reloaded with server.R errors
  - `browser` is your friend!

- Open Source free version lacks
  - Authentication
  - Multiple-threads
  - Resource allocation and monitoring