Deep Learning
Parameter Optimization

Nagiza F. Samatova, samatova@csc.ncsu.edu
Professor, Department of Computer Science
North Carolina State University

NC STATE UNIVERSITY
Department of Computer Science

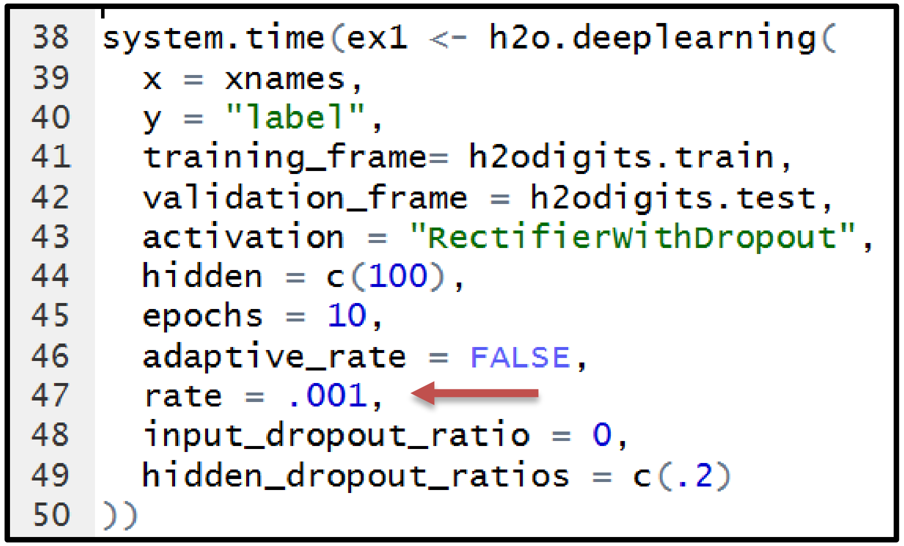
Outline

- **Model Hyperparameters**
- **Parameter Optimization Problem**
- **Parameter Optimization Strategies**
  - **Trial and error**
  - **Grid Search**
  - **Random Search**
  - **Bayesian Parameter Optimization**
- **Grid Search**
- **Random Search**

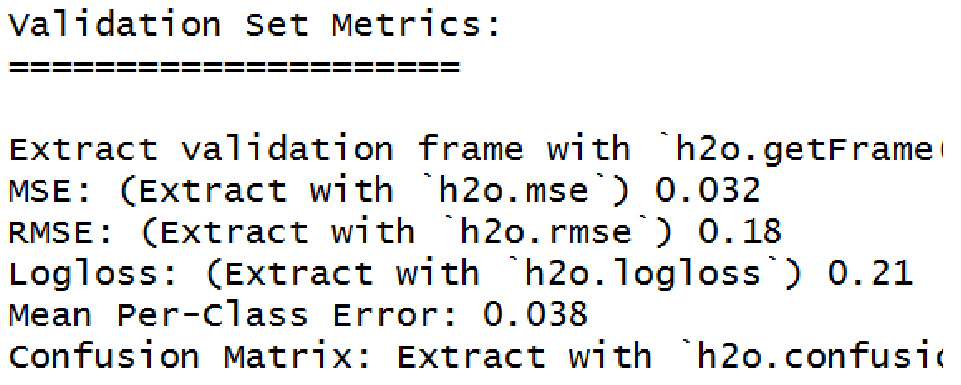
Model Parameters and Hyperparameters

- **Model Parameters**:
  - Parameters that are learned during model training
  - Examples: weights, intercept/bias parameters
- **Hyperparameters**:
  - Parameters that **must be set** at the offset and are **not optimized during model training**
- **Examples of Hyperparameters**:
  - **The choice of a model**: deep feedforward neural network, random forest, support vector machines, etc.
  - **Model architecture**: the number of hidden layers, the number of nodes in each hidden layer, types of activation functions
  - **Model Convergence and Stopping Criteria**: the number of training epochs (iterations), threshold on the difference in optimization function between two subsequent iterations, learning rate
  - **Form of regularization to avoid model overfitting**: L_1- or L_2-norm regularizer, the number of dropout neurons,

Deep NN Models: Hyperparameters

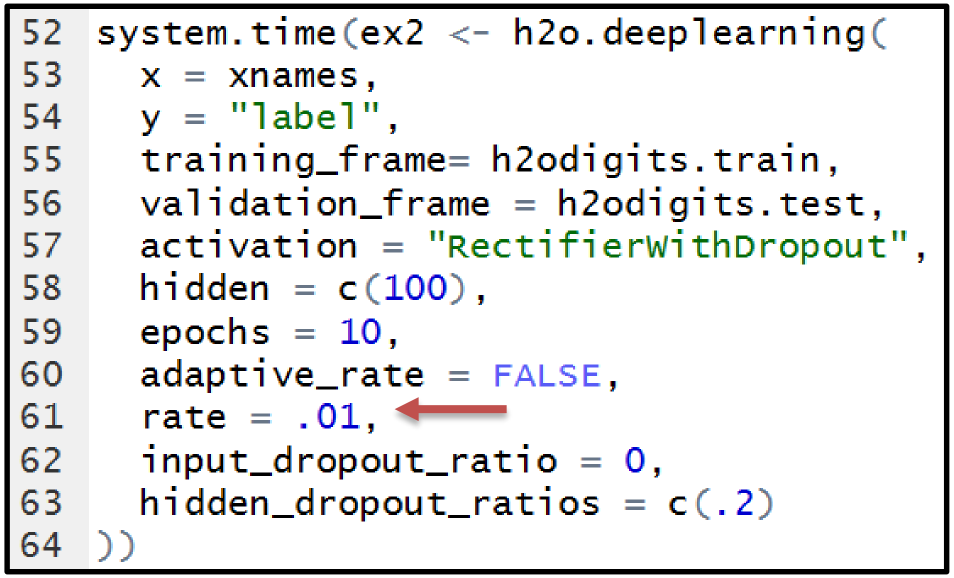


```r
system.time(ex1 <- h2o.deeplearning(
  x = xnames,
  y = "label",
  training_frame= h2odigits.train,
  validation_frame = h2odigits.test,
  activation = "RectifierWithDropout",
  hidden = c(100),
  epochs = 10,
  adaptive_rate = FALSE,
  rate = .001,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(.2)
))
```

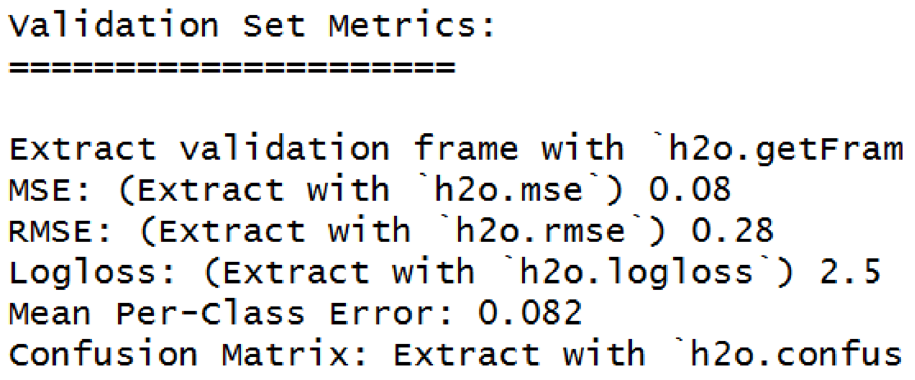


```
Validation Set Metrics:
=====================

Extract validation frame with `h2o.getFrame
MSE: (Extract with `h2o.mse`) 0.032
RMSE: (Extract with `h2o.rmse`) 0.18
Logloss: (Extract with `h2o.logloss`) 0.21
Mean Per-Class Error: 0.038
Confusion Matrix: Extract with `h2o.confusi
```

Although ex1 took longer to train, it performs substantially better on the test data than does ex2.



```r
system.time(ex2 <- h2o.deeplearning(
  x = xnames,
  y = "label",
  training_frame= h2odigits.train,
  validation_frame = h2odigits.test,
  activation = "RectifierWithDropout",
  hidden = c(100),
  epochs = 10,
  adaptive_rate = FALSE,
  rate = .01,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(.2)
))
```



```
Validation Set Metrics:
=====================

Extract validation frame with `h2o.getFram
MSE: (Extract with `h2o.mse`) 0.08
RMSE: (Extract with `h2o.rmse`) 0.28
Logloss: (Extract with `h2o.logloss`) 2.5
Mean Per-Class Error: 0.082
Confusion Matrix: Extract with `h2o.confus
```

Overarching Goal: Parameter Optimization

- **Motivation:** Some of the key reasons for poor model performance:
  - lack of the variables required for good prediction
  - not enough data to support training a complex enough model
  - **poorly tuned and optimized hyperparameters**

- **Goal: To choose the best combination of and best values for the hypermeters of the machine learning model**.
  - **Assumption:** Better hyperparameters can often improve the accuracy of a model.

The values chosen for the hyperparameters can have a dramatic impact on the accuracy and training speed of a model.

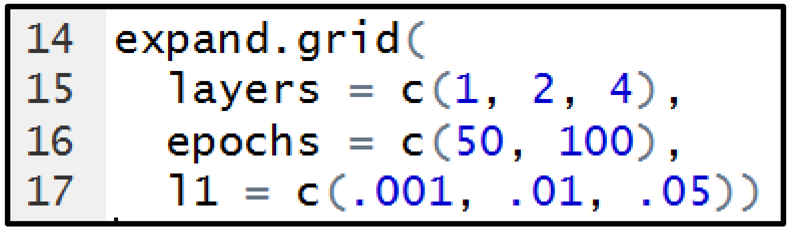
Parameter Optimization Strategies

- **Trial and error**
- **Grid Search:**
  - pros: great if there are only a **few values** for a **few parameters**
  - cons: combinatorial (brute-force exhaustive enumeration of all possible combinations)
- **Random Search:**
  - pros: searching via random sampling; no need to pre-specify all the values to try and create all possible combinations
  - cons: computationally demanding for large sample sizes
- **Bayesian Optimization (e.g., using Spearmint Python library):**
  - iteratively adjusts a number of parameters so as to minimize some objective in as few runs as possible
  - https://github.com/HIPS/Spearmint

Grid Search for Optimal Parameters

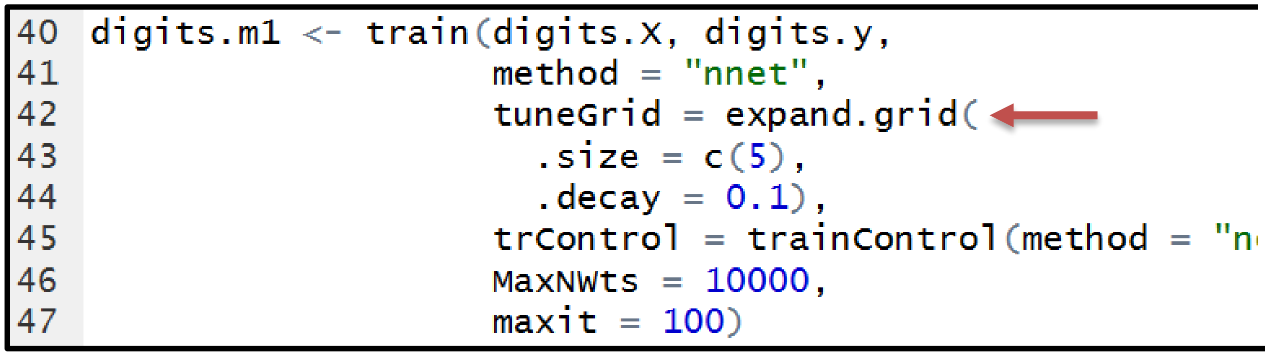
- **Basic Idea:**
  - several values for hyperparameters are specified and
  - all possible combinations of these values are tried
- To create all possible combinations in R:
  - Use **expandGrid()** function in **gridExtra** package

DeepNeuralNetwork.ParameterOptimization.R



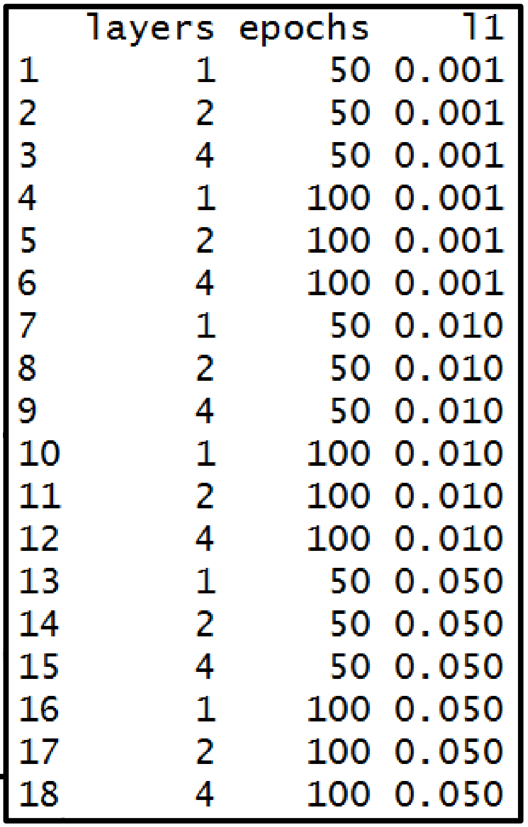
```r
expand.grid(
  layers = c(1, 2, 4),
  epochs = c(50, 100),
  l1 = c(.001, .01, .05))
```

DeepNeuralNetwork.R



```r
digits.m1 <- train(digits.X, digits.y,
                   method = "nnet",
                   tuneGrid = expand.grid(
                     .size = c(5),
                     .decay = 0.1),
                   trControl = trainControl(method = "n
                   MaxNWts = 10000,
                   maxit = 100)
```

all possible combinations



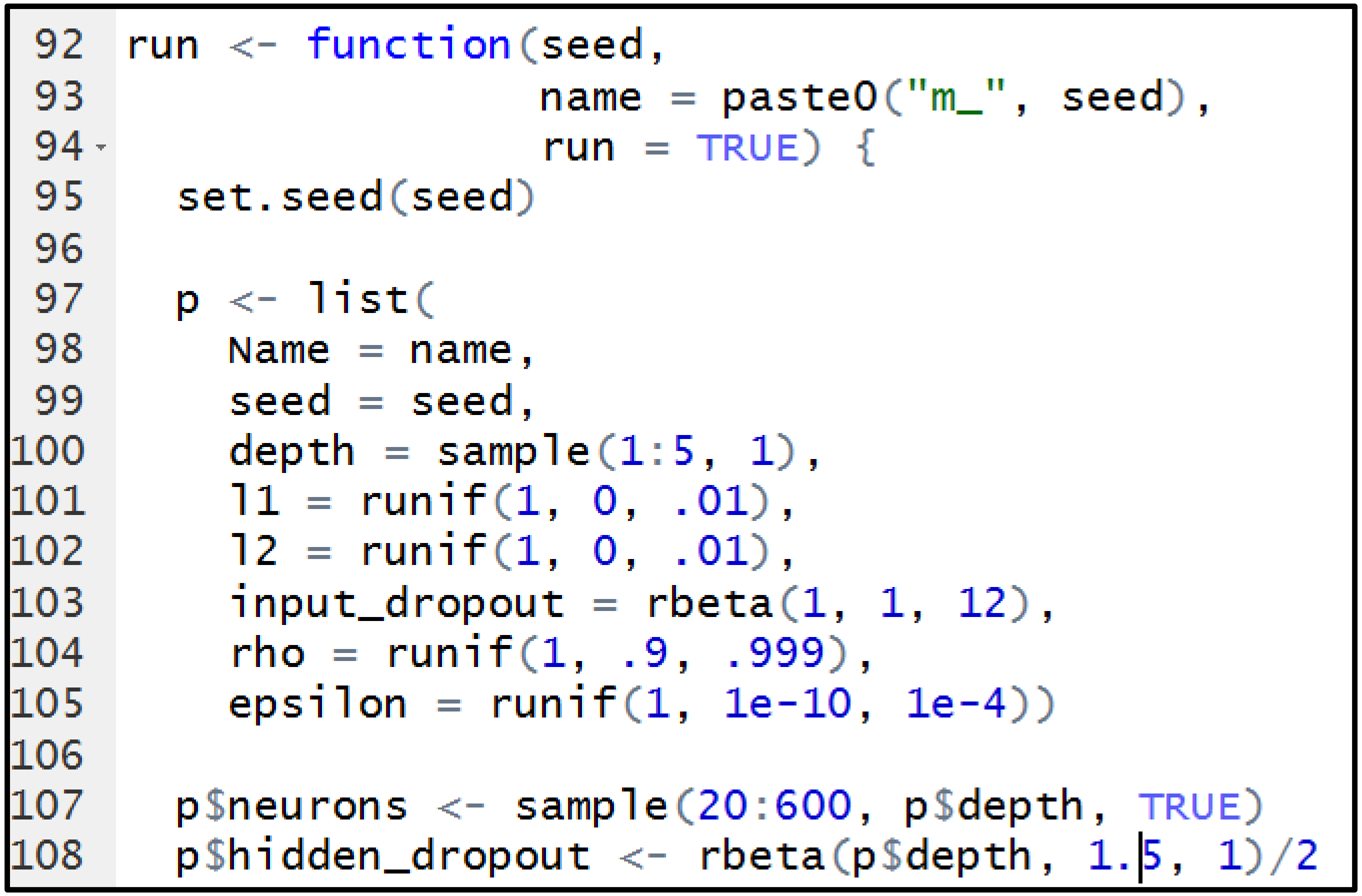
```
   layers epochs    l1
1       1     50 0.001
2       2     50 0.001
3       4     50 0.001
4       1    100 0.001
5       2    100 0.001
6       4    100 0.001
7       1     50 0.010
8       2     50 0.010
9       4     50 0.010
10      1    100 0.010
11      2    100 0.010
12      4    100 0.010
13      1     50 0.050
14      2     50 0.050
15      4     50 0.050
16      1    100 0.050
17      2    100 0.050
18      4    100 0.050
```

Random Search for Optimal Parameters

- **What to specify for random sampling:**
  - the values to randomly sample or
  - distributions to randomly draw from.
  - some limits, e.g., # of hidden layers in a range from 1 to 10
- **Examples**:
  - The **number of dropout neurons** will be drawn from **beta distribution** (20% dropouts in input layer and 50% on hidden layer): **dbeta()**
  - The depth or **number of layers** is sampled from 1 to 5
  - Adaptive learning rates (ρ and ϵ) are drawn from a **uniform distribution**: **runif()**
- **How to do random sampling:**
  - write a function that takes a seed and
  - then randomly samples a number of hyperparameters,
  - stores the sampled parameters,
  - runs the model, and
  - returns the results

Step-1: Set Up Random Search Parameters

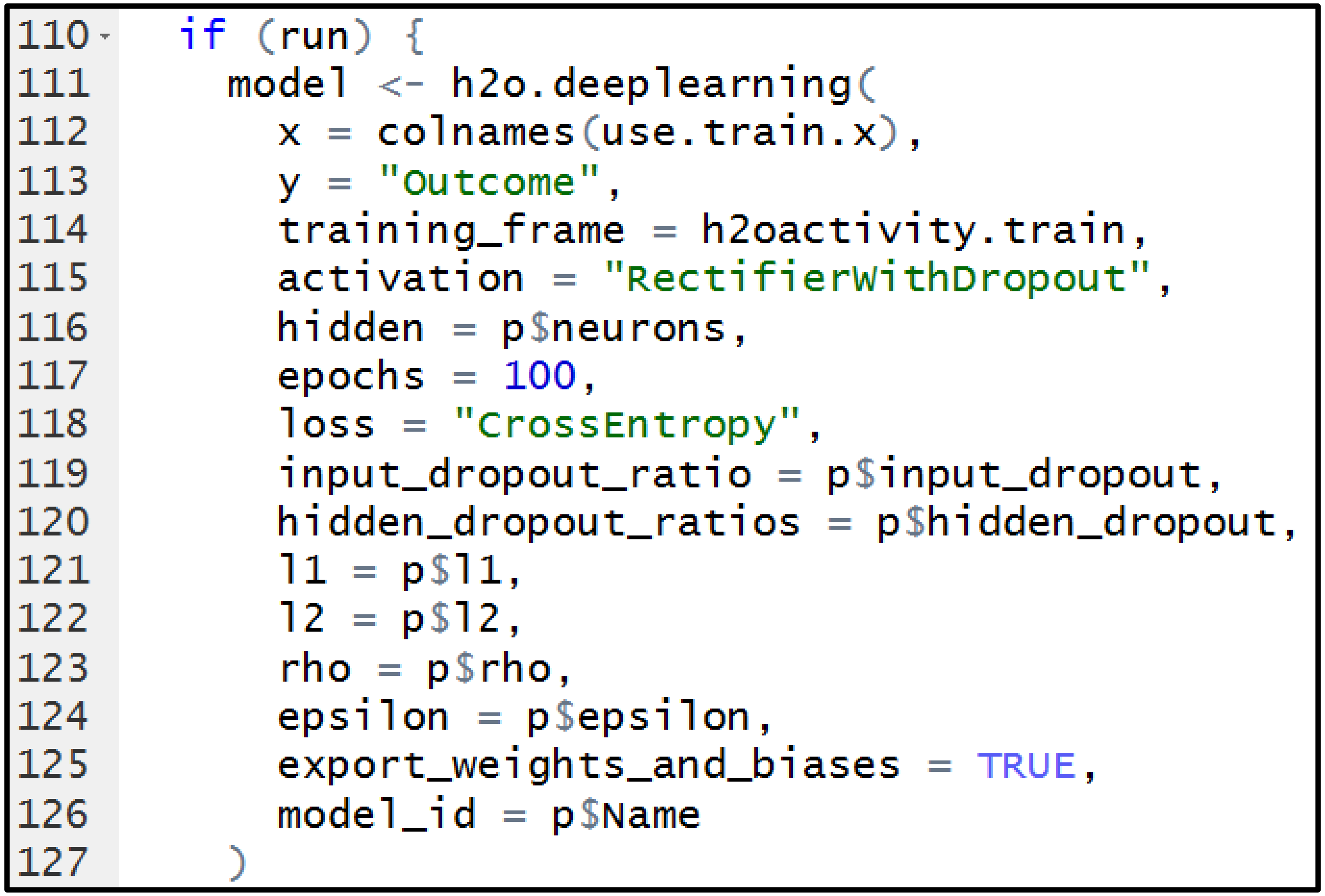
Setting Up Model Hyperparameters



```r
run <- function(seed,
                name = paste0("m_", seed),
                run = TRUE) {
  set.seed(seed)

  p <- list(
    Name = name,
    seed = seed,
    depth = sample(1:5, 1),
    l1 = runif(1, 0, .01),
    l2 = runif(1, 0, .01),
    input_dropout = rbeta(1, 1, 12),
    rho = runif(1, .9, .999),
    epsilon = runif(1, 1e-10, 1e-4))

  p$neurons <- sample(20:600, p$depth, TRUE)
  p$hidden_dropout <- rbeta(p$depth, 1.5, 1)/2
```

Step-2: Build Models for these Parameters

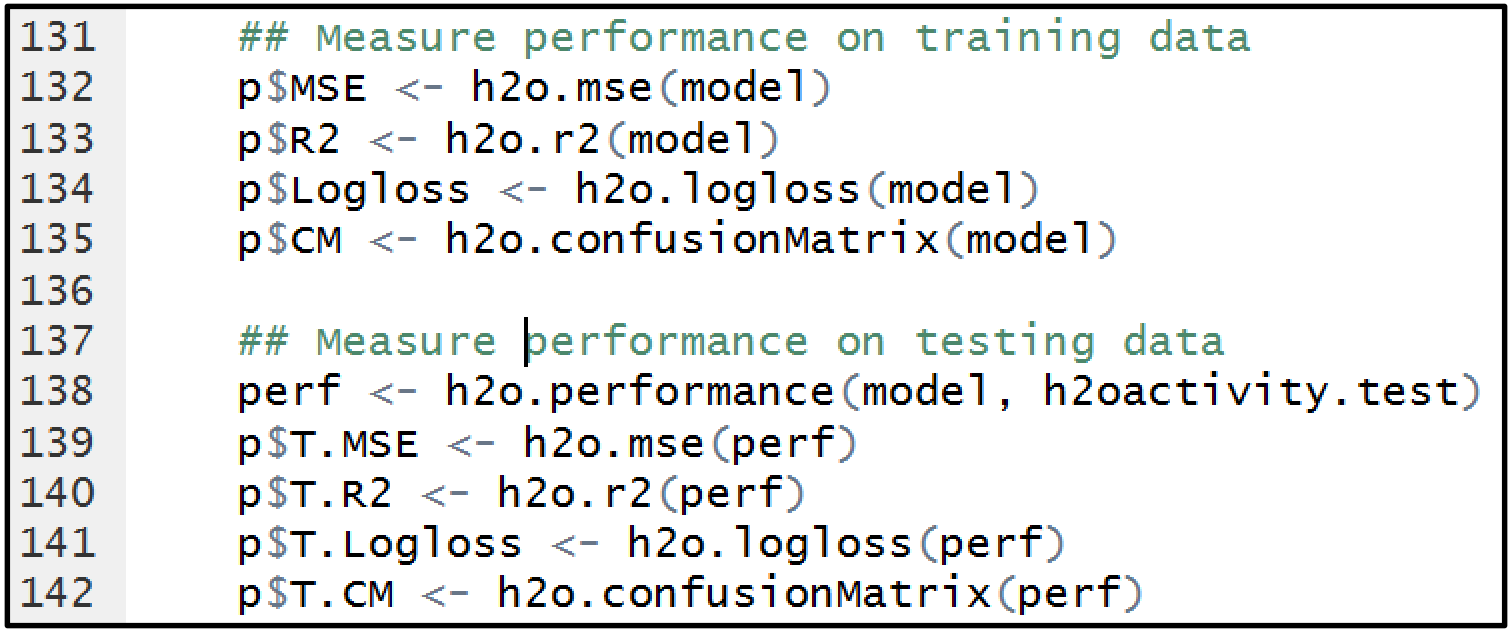
Building Models for Different Model Hyperparameters



```
  if (run) {
    model <- h2o.deeplearning(
      x = colnames(use.train.x),
      y = "Outcome",
      training_frame = h2oactivity.train,
      activation = "RectifierWithDropout",
      hidden = p$neurons,
      epochs = 100,
      loss = "CrossEntropy",
      input_dropout_ratio = p$input_dropout,
      hidden_dropout_ratios = p$hidden_dropout,
      l1 = p$l1,
      l2 = p$l2,
      rho = p$rho,
      epsilon = p$epsilon,
      export_weights_and_biases = TRUE,
      model_id = p$Name
    )
```

Step-3: Measure Models' Performance

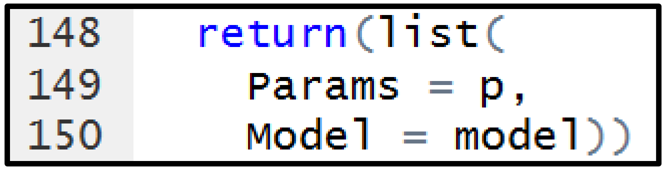
Measure Model Performance for Different Model Hyperparameters



```
    ## Measure performance on training data
    p$MSE <- h2o.mse(model)
    p$R2 <- h2o.r2(model)
    p$Logloss <- h2o.logloss(model)
    p$CM <- h2o.confusionMatrix(model)

    ## Measure performance on testing data
    perf <- h2o.performance(model, h2oactivity.test)
    p$T.MSE <- h2o.mse(perf)
    p$T.R2 <- h2o.r2(perf)
    p$T.Logloss <- h2o.logloss(perf)
    p$T.CM <- h2o.confusionMatrix(perf)
```

Step-4: Return Parameters, Model & Results

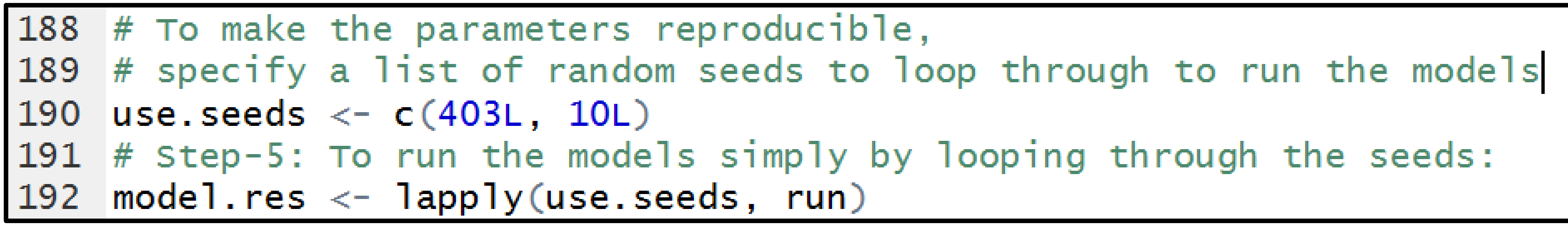
Return Model Results for Different Model Hyperparameters



```
  return(list(
    Params = p,
    Model = model))
```

Step-5: Generate Results for Different Seeds

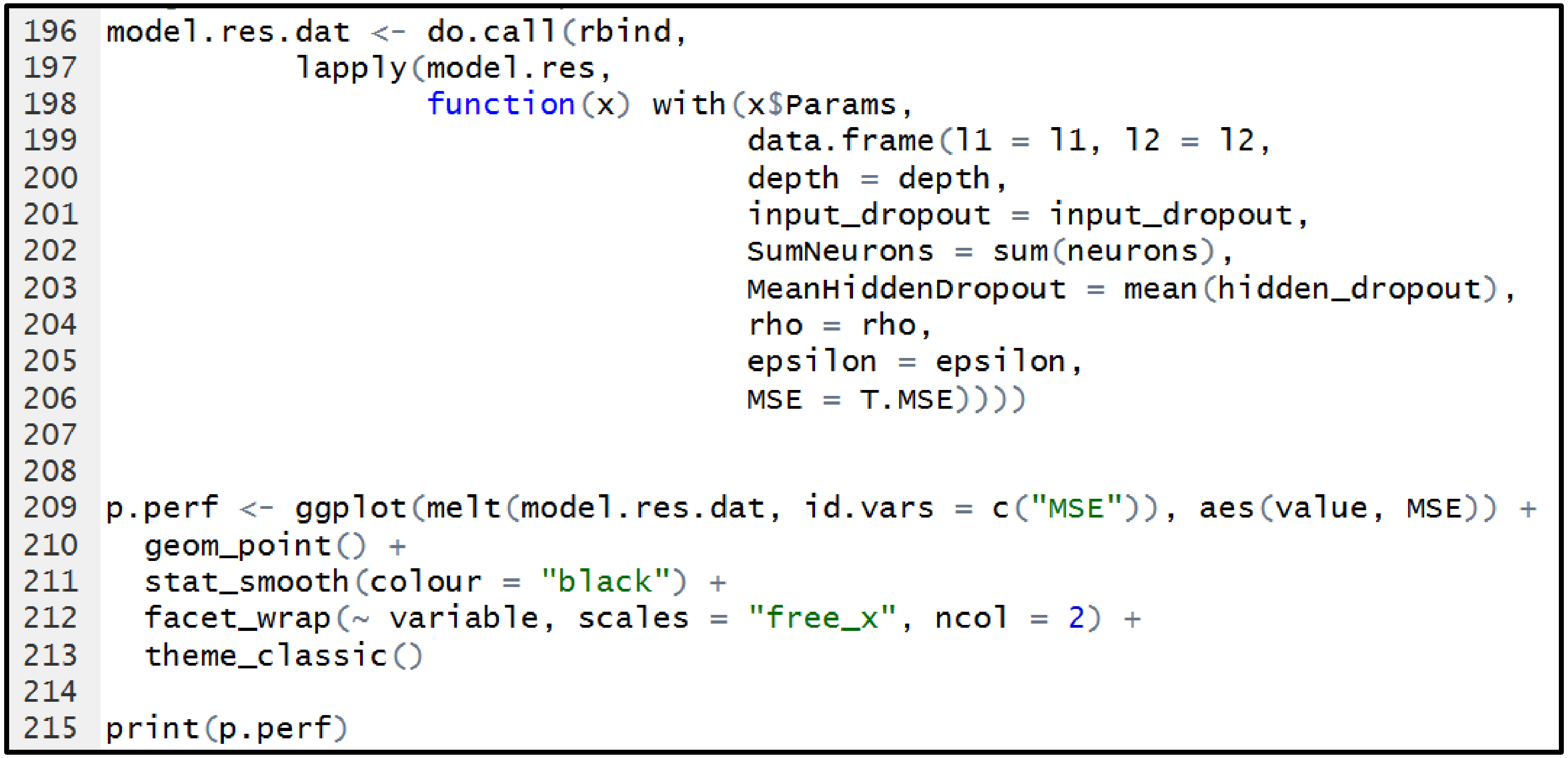
To make results reproducible, use different seeds:



```r
# To make the parameters reproducible,
# specify a list of random seeds to loop through to run the models
use.seeds <- c(403L, 10L)
# Step-5: To run the models simply by looping through the seeds:
model.res <- lapply(use.seeds, run)
```

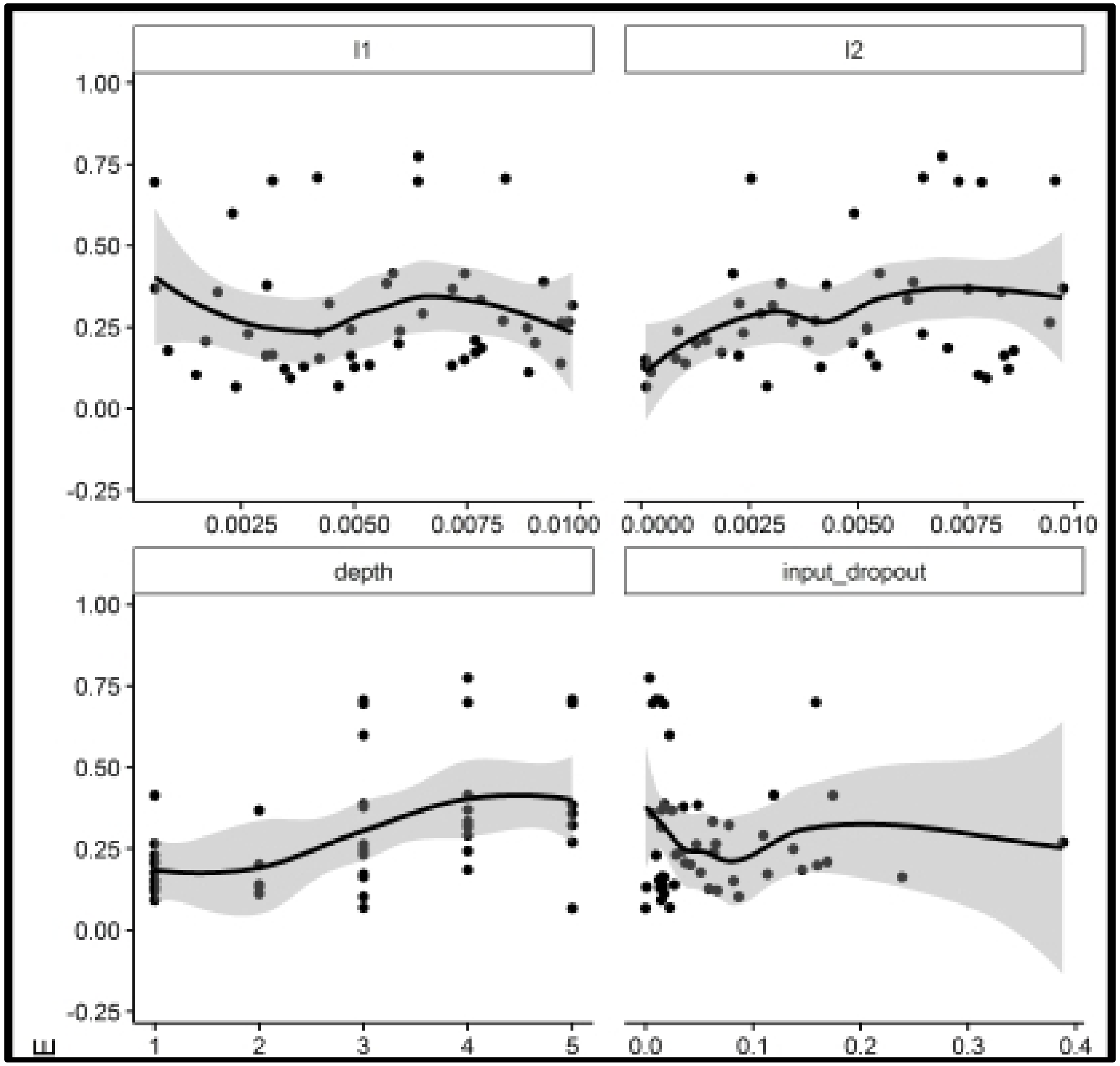
Step-6: Examine MSE for Diff. Parameters

Plot the mean squared error (MSE) against the different parameters

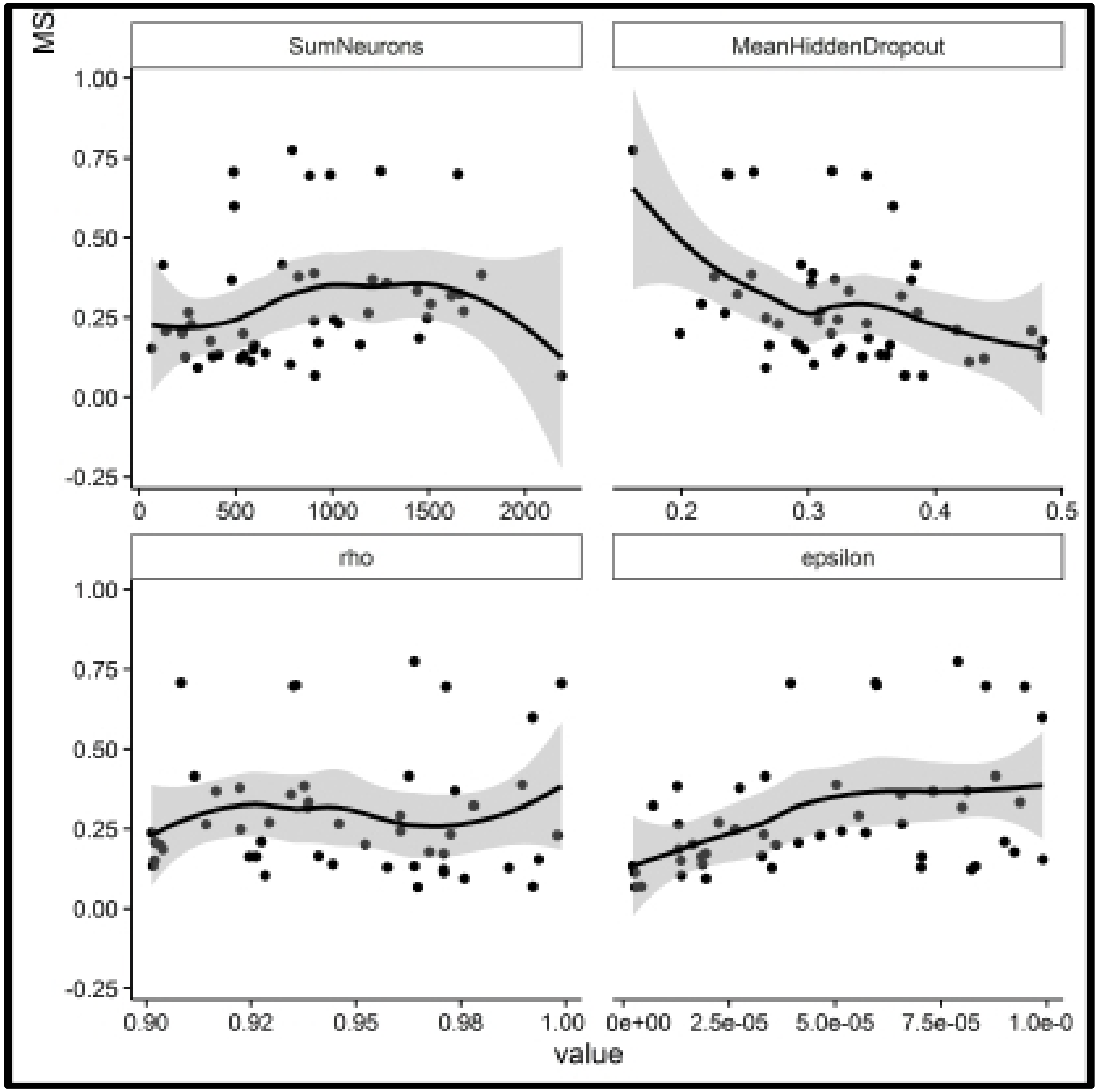


```r
model.res.dat <- do.call(rbind,
          lapply(model.res,
                 function(x) with(x$Params,
                                  data.frame(l1 = l1, l2 = l2,
                                  depth = depth,
                                  input_dropout = input_dropout,
                                  SumNeurons = sum(neurons),
                                  MeanHiddenDropout = mean(hidden_dropout),
                                  rho = rho,
                                  epsilon = epsilon,
                                  MSE = T.MSE))))


p.perf <- ggplot(melt(model.res.dat, id.vars = c("MSE")), aes(value, MSE)) +
  geom_point() +
  stat_smooth(colour = "black") +
  facet_wrap(~ variable, scales = "free_x", ncol = 2) +
  theme_classic()

print(p.perf)
```

Step-6: MSE Plots for Different Seeds

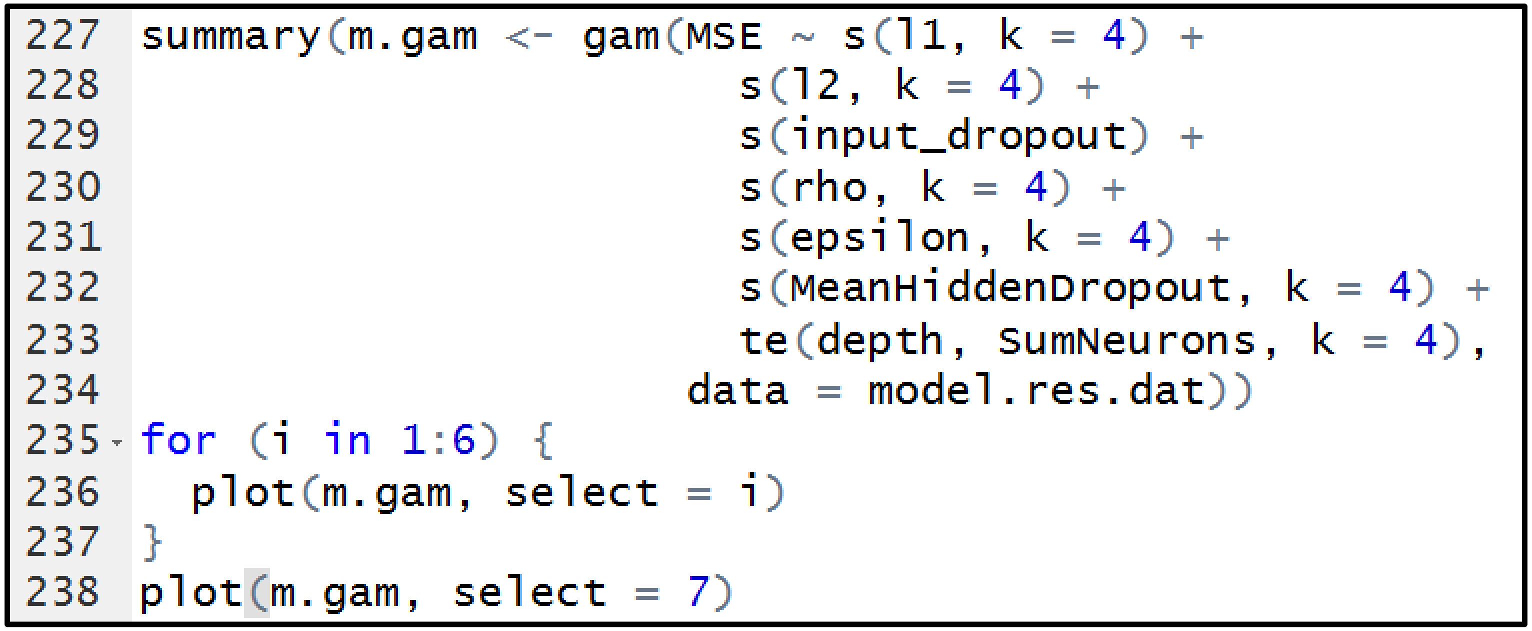


Step-6: MSE Plots for Different Seeds



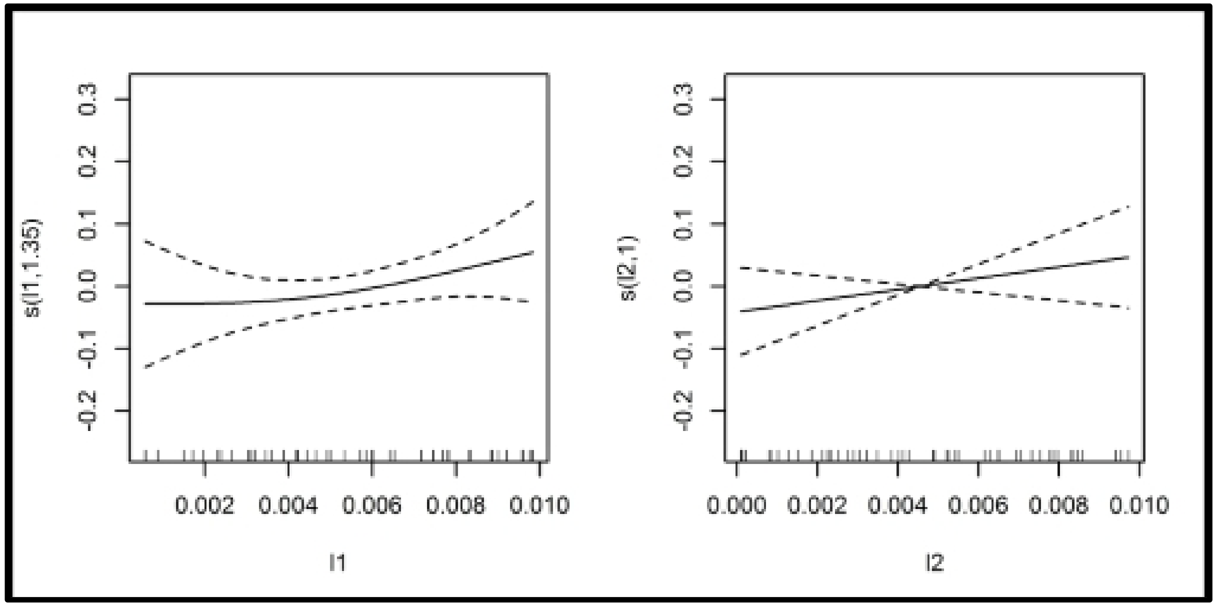
Step-7: Multivariate Parameter Relationships

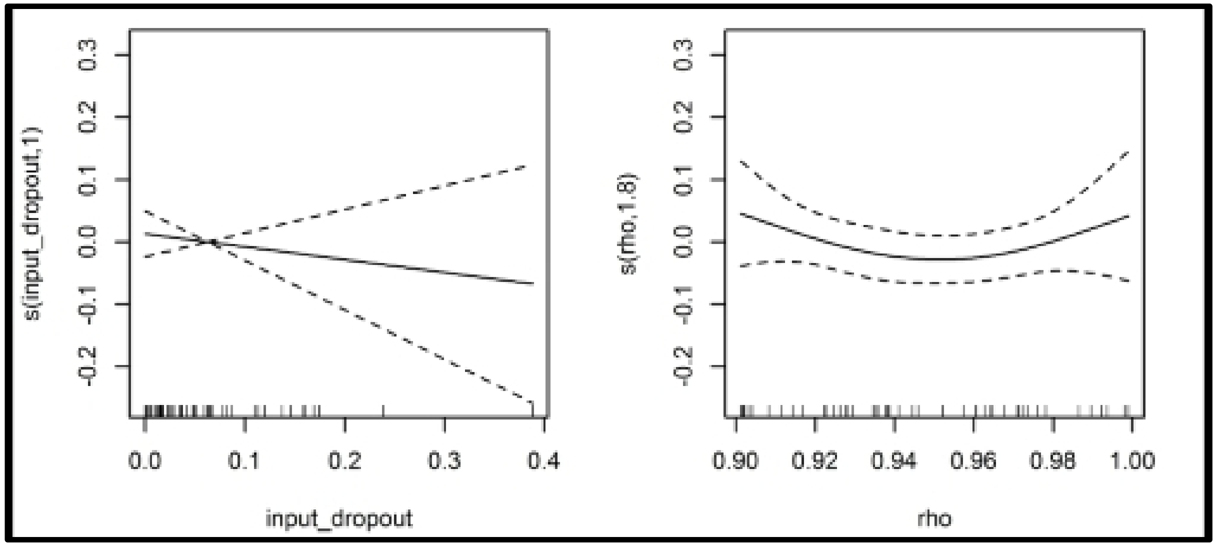
Examine multivariate parameter relationships w.r.t. MSE



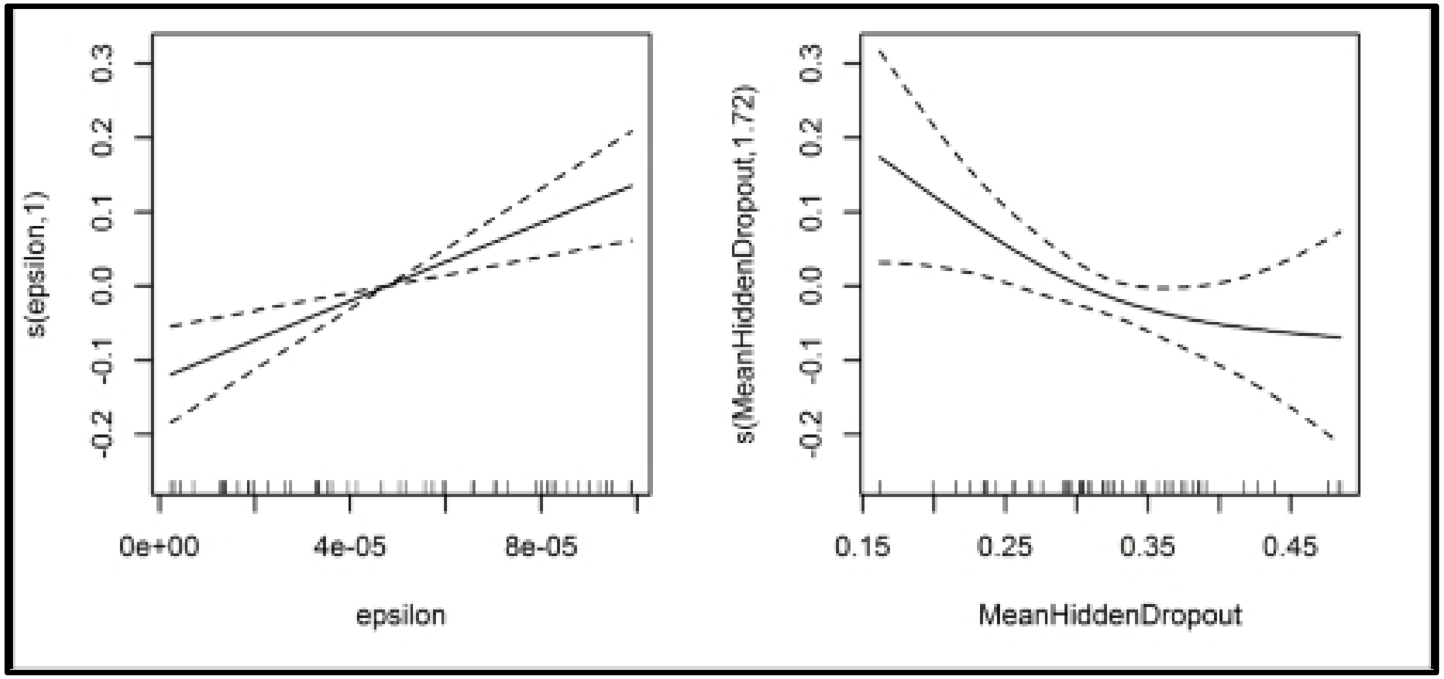
```r
summary(m.gam <- gam(MSE ~ s(l1, k = 4) +
                         s(l2, k = 4) +
                         s(input_dropout) +
                         s(rho, k = 4) +
                         s(epsilon, k = 4) +
                         s(MeanHiddenDropout, k = 4) +
                         te(depth, SumNeurons, k = 4),
                     data = model.res.dat))
for (i in 1:6) {
  plot(m.gam, select = i)
}
plot(m.gam, select = 7)
```

Step-7: MSE Multivariate Relation Plots

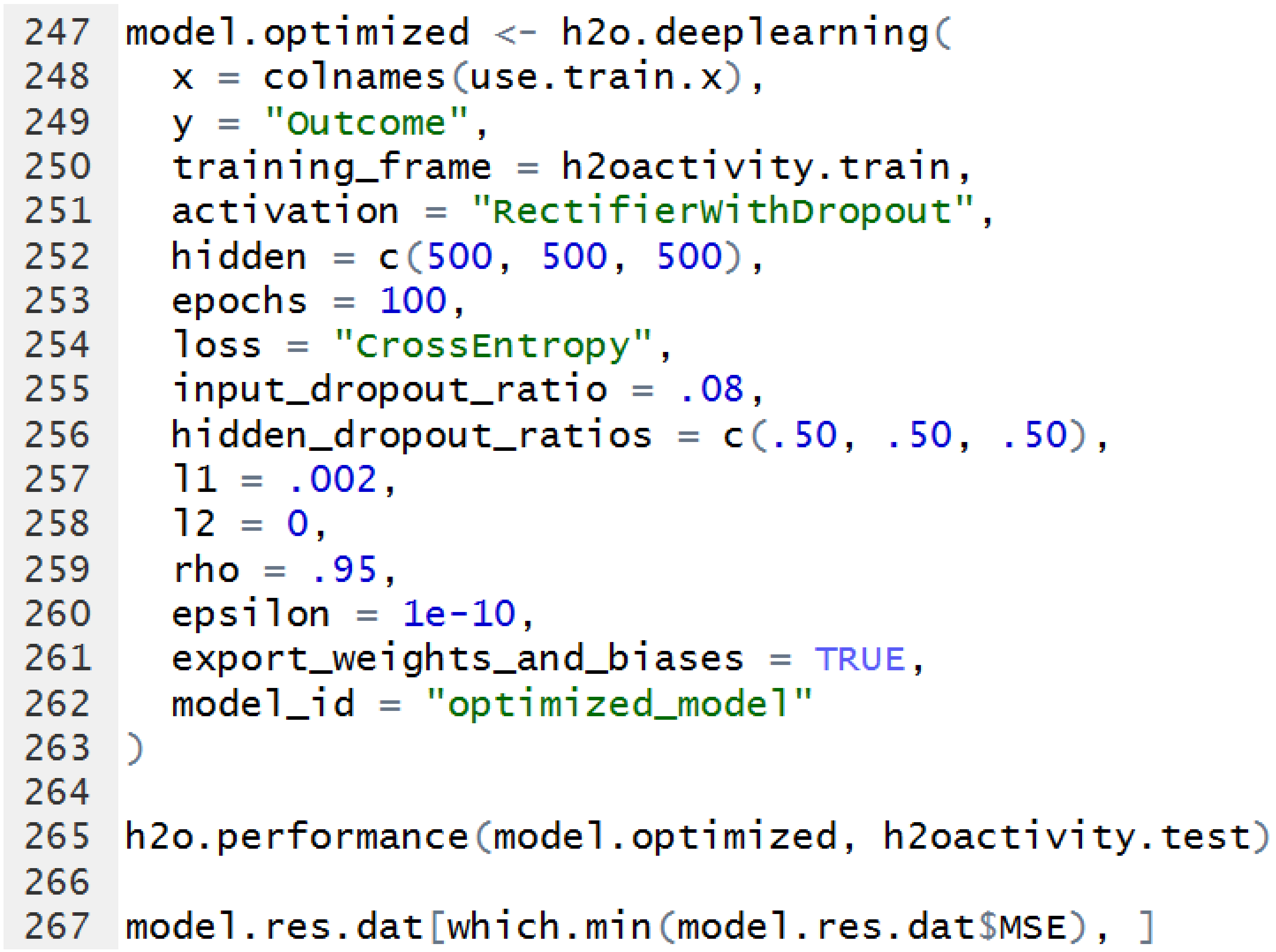




Step-7: MSE Multivariate Relation Plots



Step-8: Choose Optimal Model Parameters



```r
model.optimized <- h2o.deeplearning(
  x = colnames(use.train.x),
  y = "Outcome",
  training_frame = h2oactivity.train,
  activation = "RectifierWithDropout",
  hidden = c(500, 500, 500),
  epochs = 100,
  loss = "CrossEntropy",
  input_dropout_ratio = .08,
  hidden_dropout_ratios = c(.50, .50, .50),
  l1 = .002,
  l2 = 0,
  rho = .95,
  epsilon = 1e-10,
  export_weights_and_biases = TRUE,
  model_id = "optimized_model"
)

h2o.performance(model.optimized, h2oactivity.test)

model.res.dat[which.min(model.res.dat$MSE), ]
```

References

- **Chapter 19, Design and Analysis of Machine Learning Experiments**
- **Bengio, Y.** (2012), **Section 3**, **Hyper-Parameters** in Practical Recommendations for Gradient-Based Training of Deep Architectures. In Neural Networks: Tricks of the Trade (pp. 437-478). Springer Berlin Heidelberg. (Also on the arXiv: http://arxiv.org/pdf/1206.5533.pdf)
- **Zeiler, M. D.** (2012), **ADADELTA**: An Adaptive Learning Rate Method. arXiv:1212.5701
- Bayesian optimization of hyperparameters, available online:
  - https://github.com/HIPS/Spearmint
  - see references on github for Spearmint library
- Joshua F. Wiley , R Deep Learning Essentials