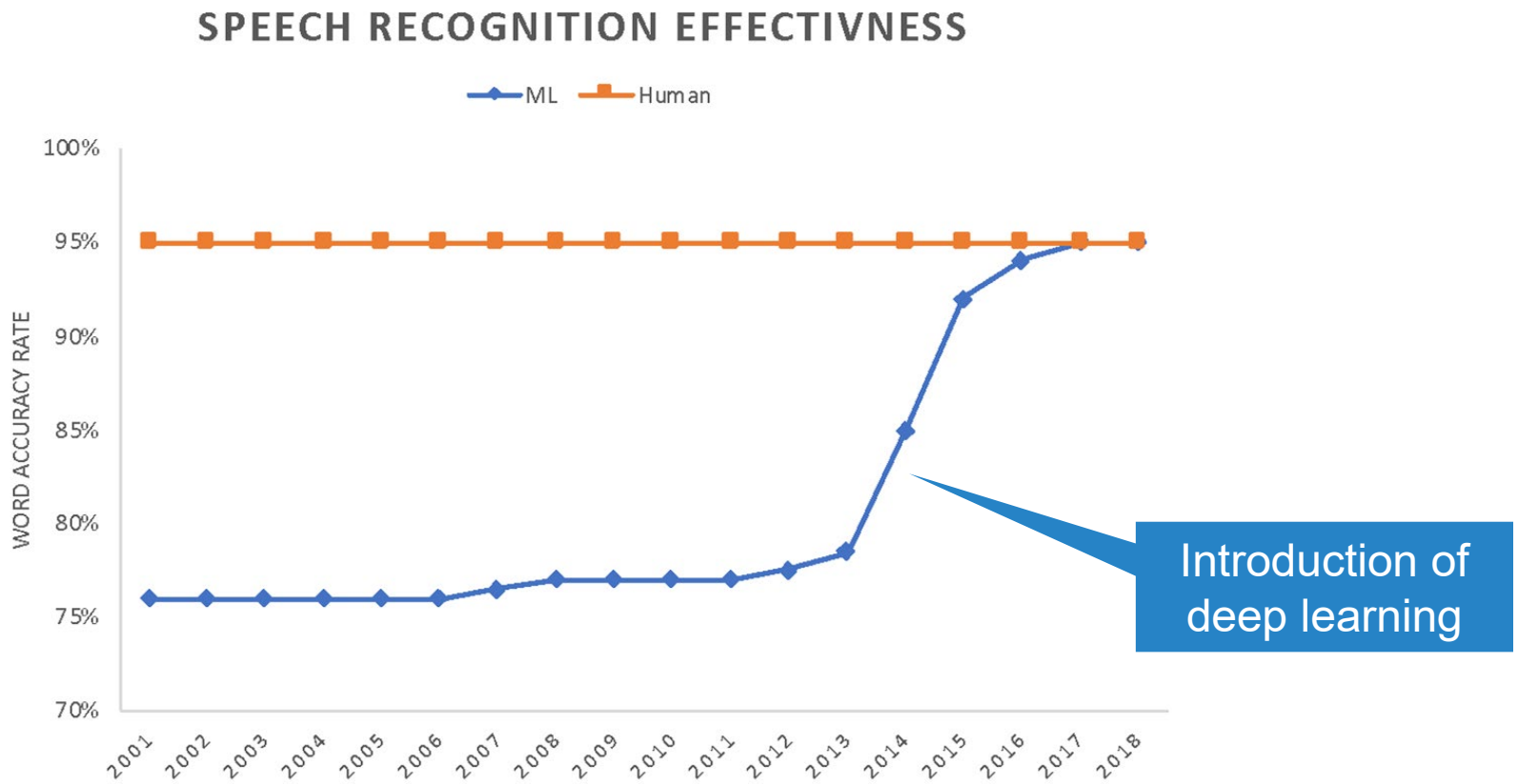
Deep Learning Overview

School of Information Studies
Syracuse University

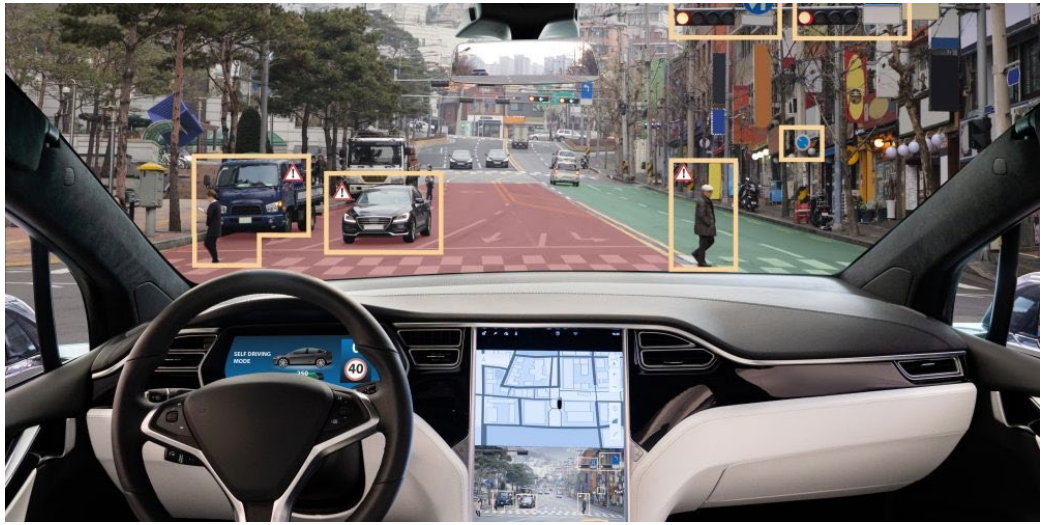
Deep Learning Impact: Speech Recognition



Source: https://www.slideshare.net/kleinerperkins/internet-trends-2017-report
https://sonix.ai/history-of-speech-recognition

Example Deep Learning Applications

Self-driving cars
Voice search and voice-activated assistants
Automatically adding sounds to silent movies
Automatic machine translation
Automatic text generation
Automatic handwriting generation
Image recognition
Automatic image caption generation
Predicting earthquakes
Brain cancer detection
Energy market price forecasting

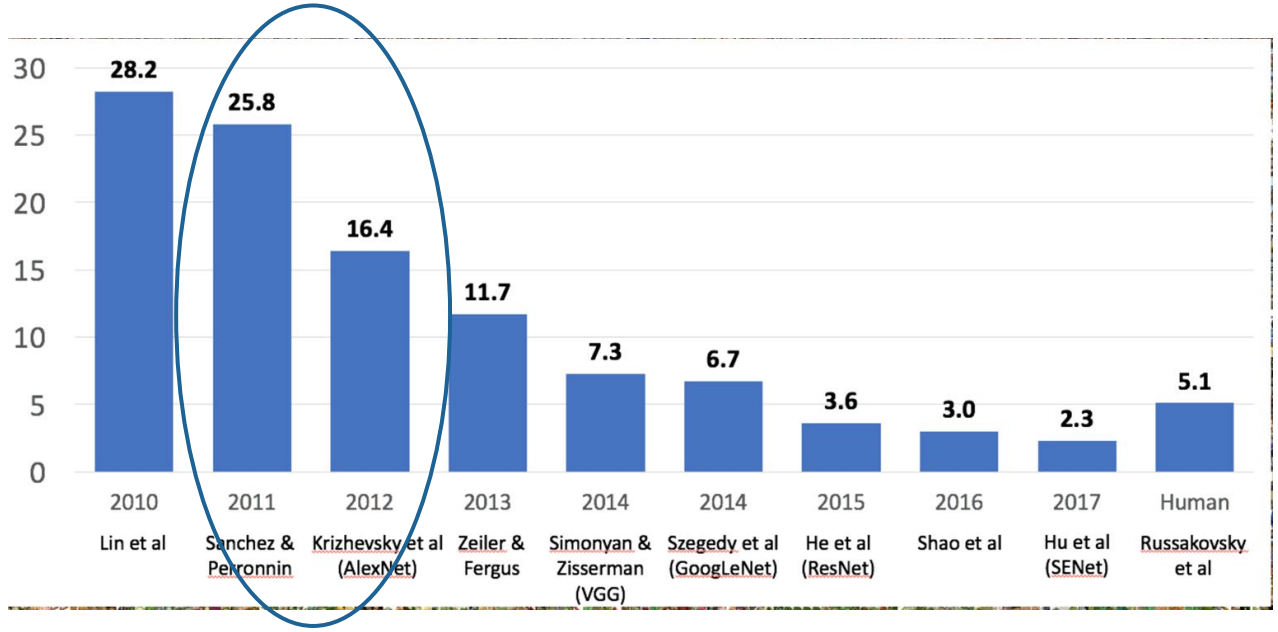


https://medium.com/breathe-publication/top-15-deep-learning-applications-that-will-rule-the-world-in-2018-and-beyond-7c6130c43b01

Deep Learning Impact: Image Recognition

ImageNet large scale visual recognition challenge

- When deep learning was first used (2012), the deep neural network was 41% better than the previous best solution.



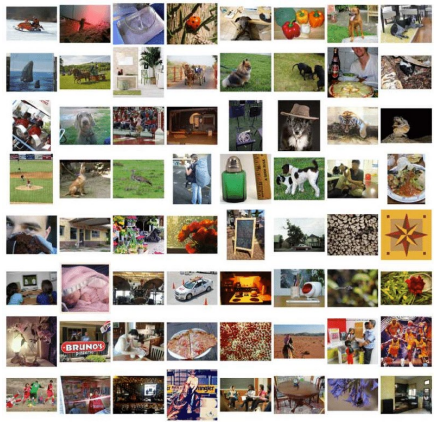
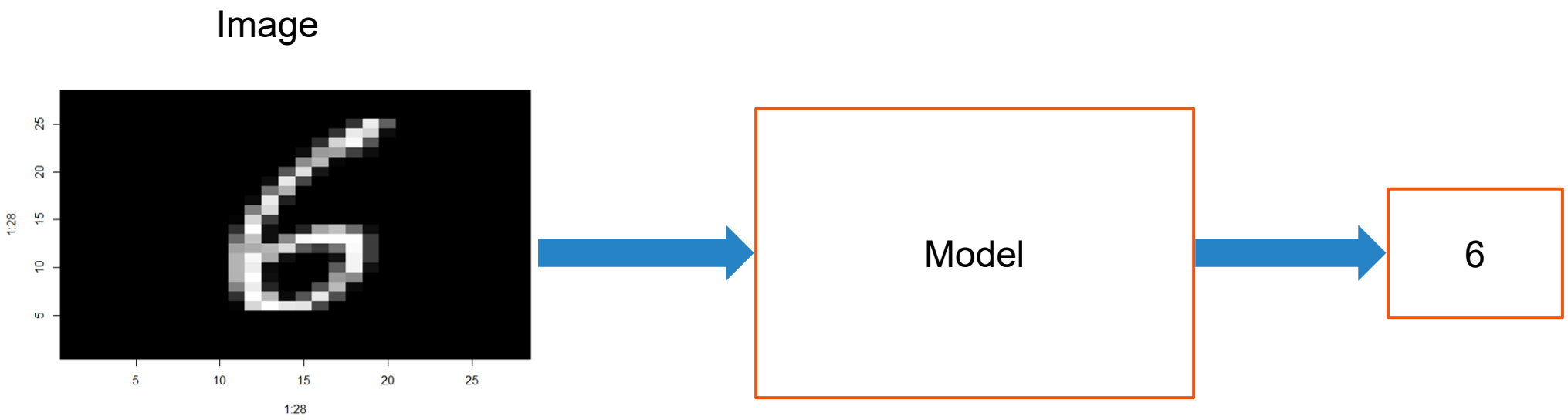


Image credit: https://www.researchgate.net/figure/Examples-in-the-ImageNet-dataset_fig7_314646236

ImageNet information: https://ruder.io/nlp-imagenet/
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture01.pdf

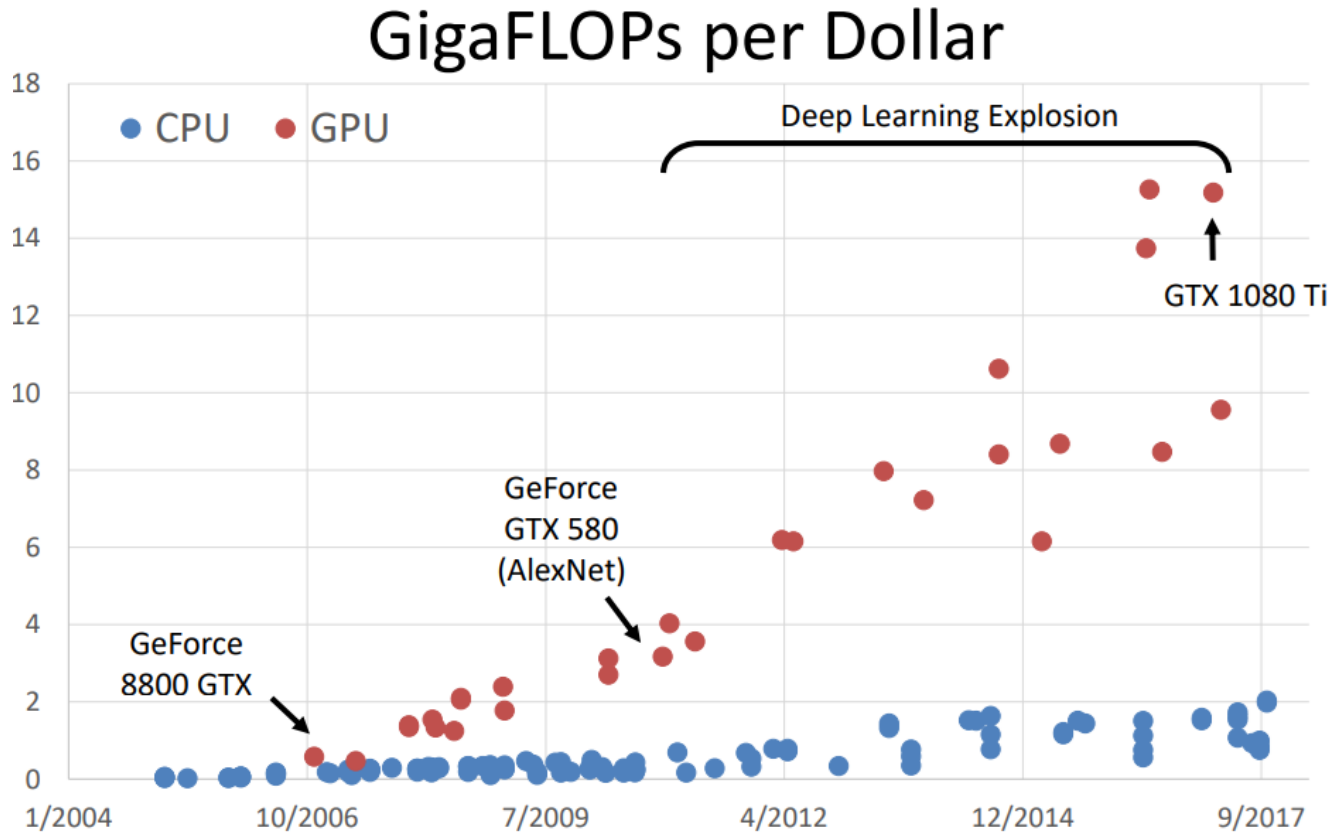
Typical Deep Learning Application

Predict what is the digit in an image



Why Deep Learning Started to be Used?

Low-Cost Computing and Storage



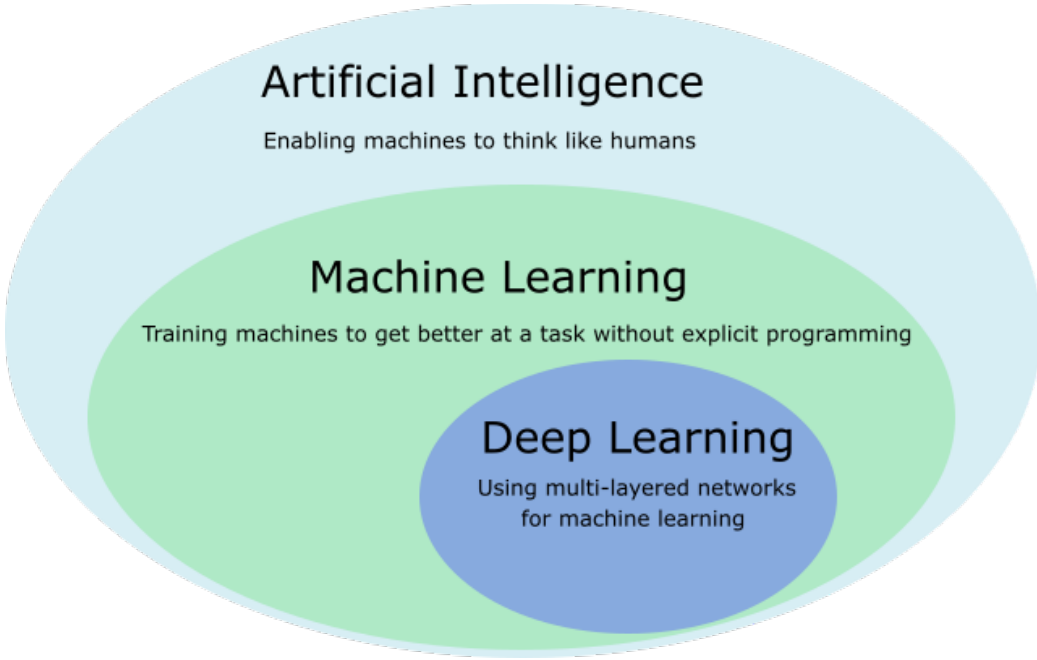
Reference: http://cs229.stanford.edu/materials/CS229-DeepLearning.pdf

What Is Deep Learning?

It's an artificial intelligence capability that uses machine learning to imitate the human brain.

Creates a network of "neurons" capable of learning from data. Also known as:

- Deep neural learning
- Deep neural network



Reference
https://www.datamation.com/artificial-intelligence/ai-vs-machine-learning-vs-deep-learning.html

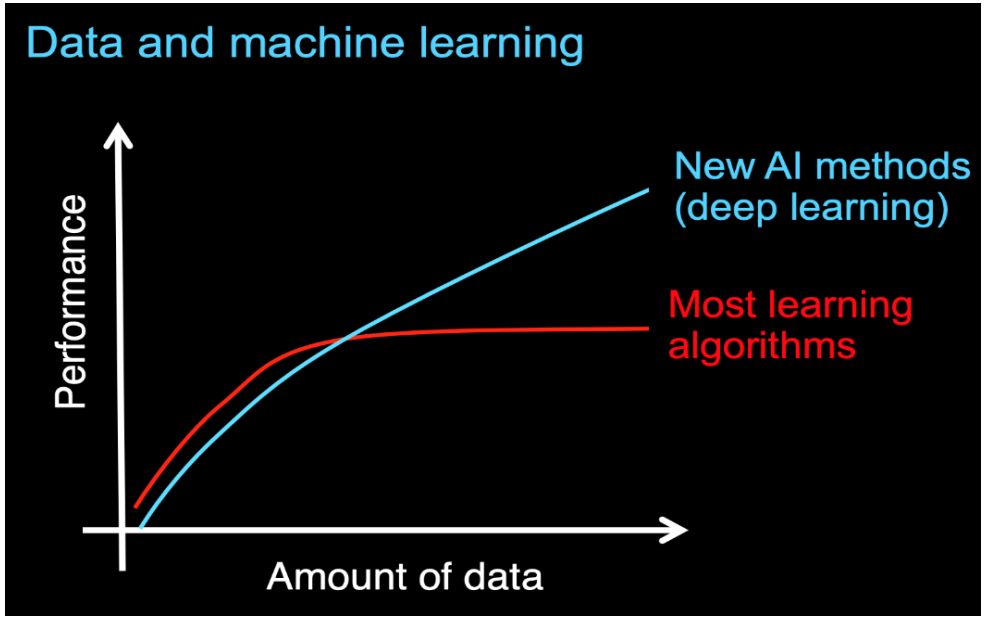
Why Is Deep Learning Useful?

Deep learning

- Utilizes large amounts of training data
- Provides a **flexible** framework for representing world, visual, and linguistic information

Feature engineering

- Manually designed features are often **over-specified, incomplete,** and take a **long time to design** and validate.
- Machine learned features are **easy to adapt, fast** to learn.



Reference
http://cs229.stanford.edu/materials/CS229-DeepLearning.pdf

Deep Learning vs. SVMs

SVMs are nonparametric models

- **Complexity grows as the training size increases** (i.e., training the model can be expensive computationally).
- In a complex data set worst-case scenario, we can end up with as many support vectors as we have samples in the training set.

SVMs and other simpler models are typically preferred for relatively small data sets with fewer outliers.

Deep learning needs relatively large data sets (and infrastructure) to train the model.

Reference:
Raschka, S. (2016, April 22). When Does Deep Learning Work Better Than SVMs or Random Forests? Retrieved from https://www.kdnuggets.com/2016/04/deep-learning-vs-svm-random-forest.html

Basic Deep Learning Vocabulary

Epoch: an iteration of learning (one pass of the training data)

Accuracy: percentage of correct predictions (e.g., based on confusion matrix)

Loss: how inaccurate is the prediction by looking at prediction percentage (predicting 90% for the correct choice is better than predicting 70%)

Tensor: a generalization of vectors and matrices (i.e., a multidimensional array)

Two main parameters control the architecture (or topology) of the deep learning network:

1. The number of **layers** in the network
2. The number of **neurons (nodes)** in each layer

Neuron/Perceptron Example

Which car should I buy?

Factor number	Factor description	Factor weight
X1	Is the miles per gallon (mpg) above 30 mpg?	W1 = 3 (high MPG is important)
X2	Is the horsepower above 200?	W2 = 1 (high horsepower is somewhat important)
X3	Is the transmission type auto?	W3 = 5 (the car really needs to be an auto transmission)

Each car will have a score (output) from the neuron

The neuron has 3 inputs (factors) to determine the output

A car with 32 mpg (X1=1), 190 hp (X2=0), and auto transmission (X3=1), will have an output of 8:

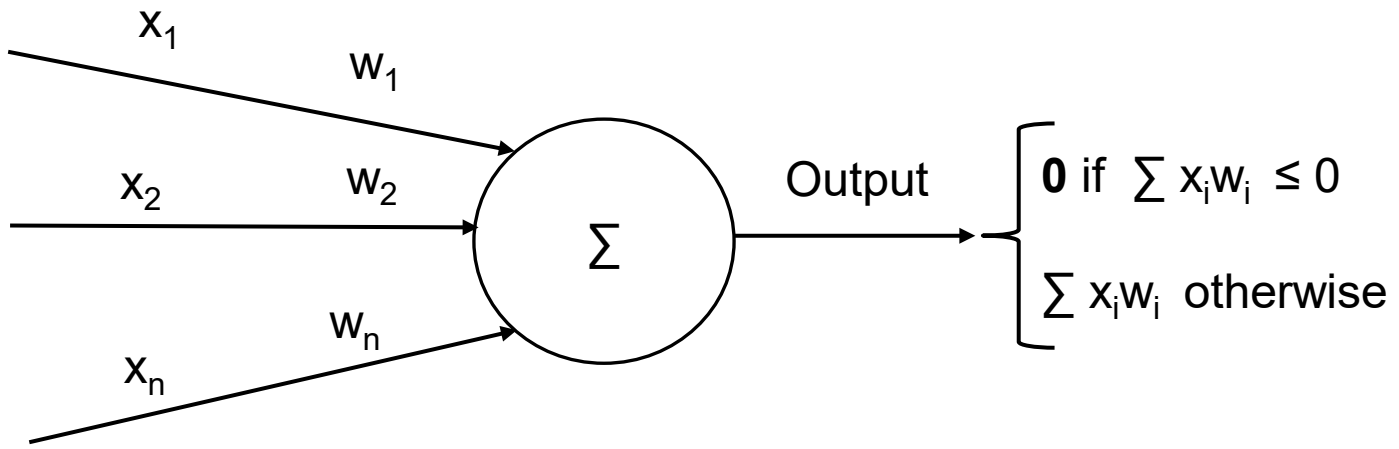
Output = X1*W1 + X2*W2 + X3*W3

= 3 + 0 + 5

Node/Neuron/Perceptron

A computational unit that has one or more weighted input connections, a transfer/activation function that combines the inputs in some way, and an output connection.

An example ***activation function*** for a neuron
Rectified linear unit (ReLU) − the value or 0 if negative :



The Impact of More Neurons

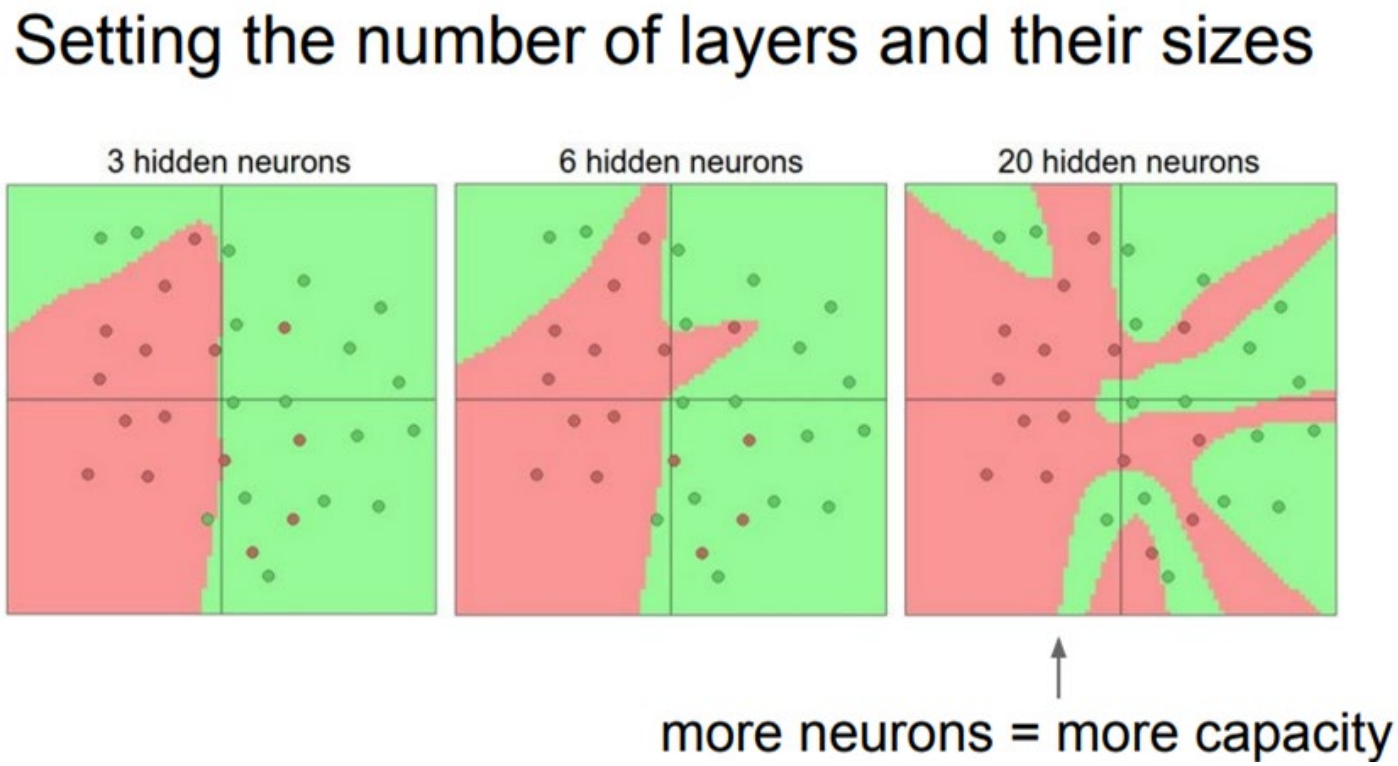


Image credit: *http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf*

Why Multiple Layers of Neurons?

A *single-layer* neural network can only be used to represent *linearly separable* functions.

- Problems such as when the two classes in a classification problem can be neatly separated by a line

However, most problems are not linearly separable.

A *multilayer* network can be used to represent *convex* regions.

- The networks can learn to draw shapes around examples in a high-dimensional space that can separate and classify the regions
- This overcomes the limitation of linear separability

Example: Layers Used for Face Recognition

Layers progressively add more insight as the data passes from input layers toward output layers.

First layer: recognizes edges

Second layer: recognizes facial features like a nose or an ear

Third layer: recognizes faces

Fourth layer: the full face is recognized

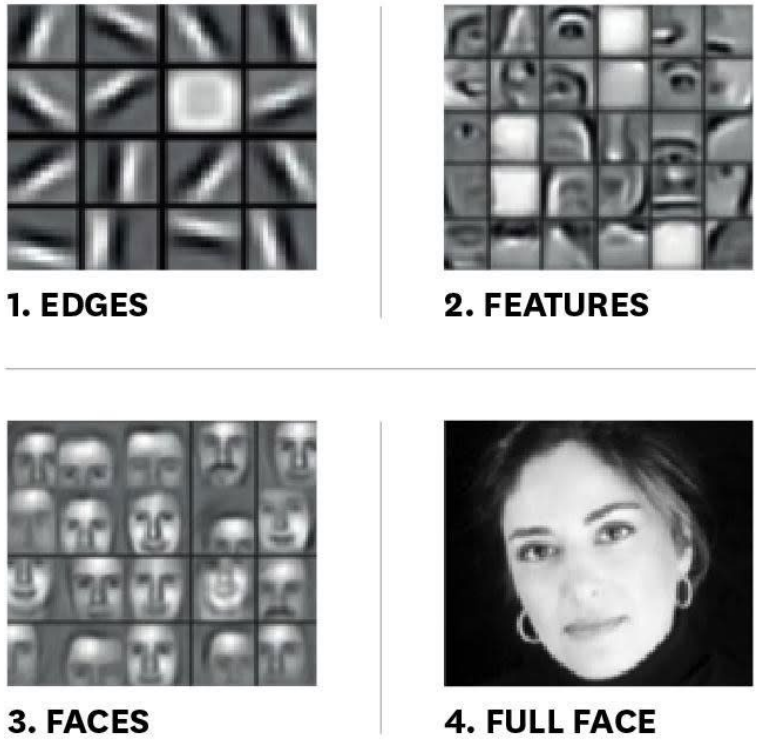
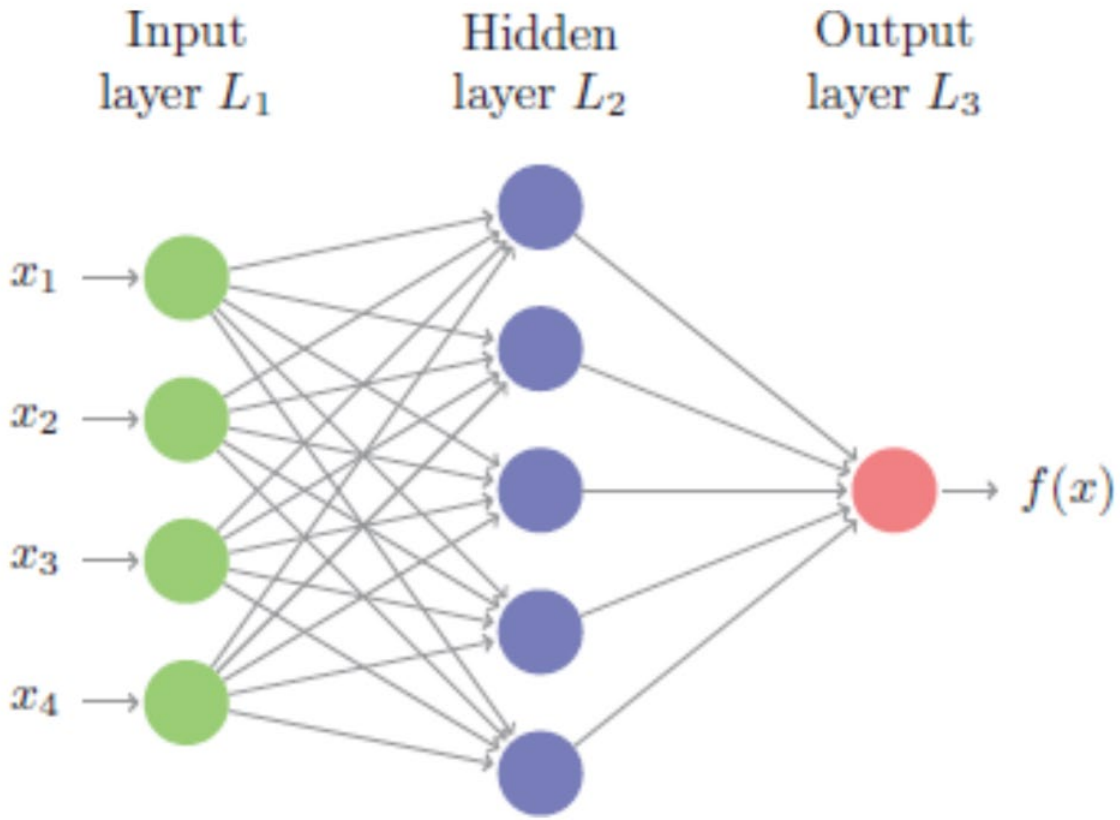
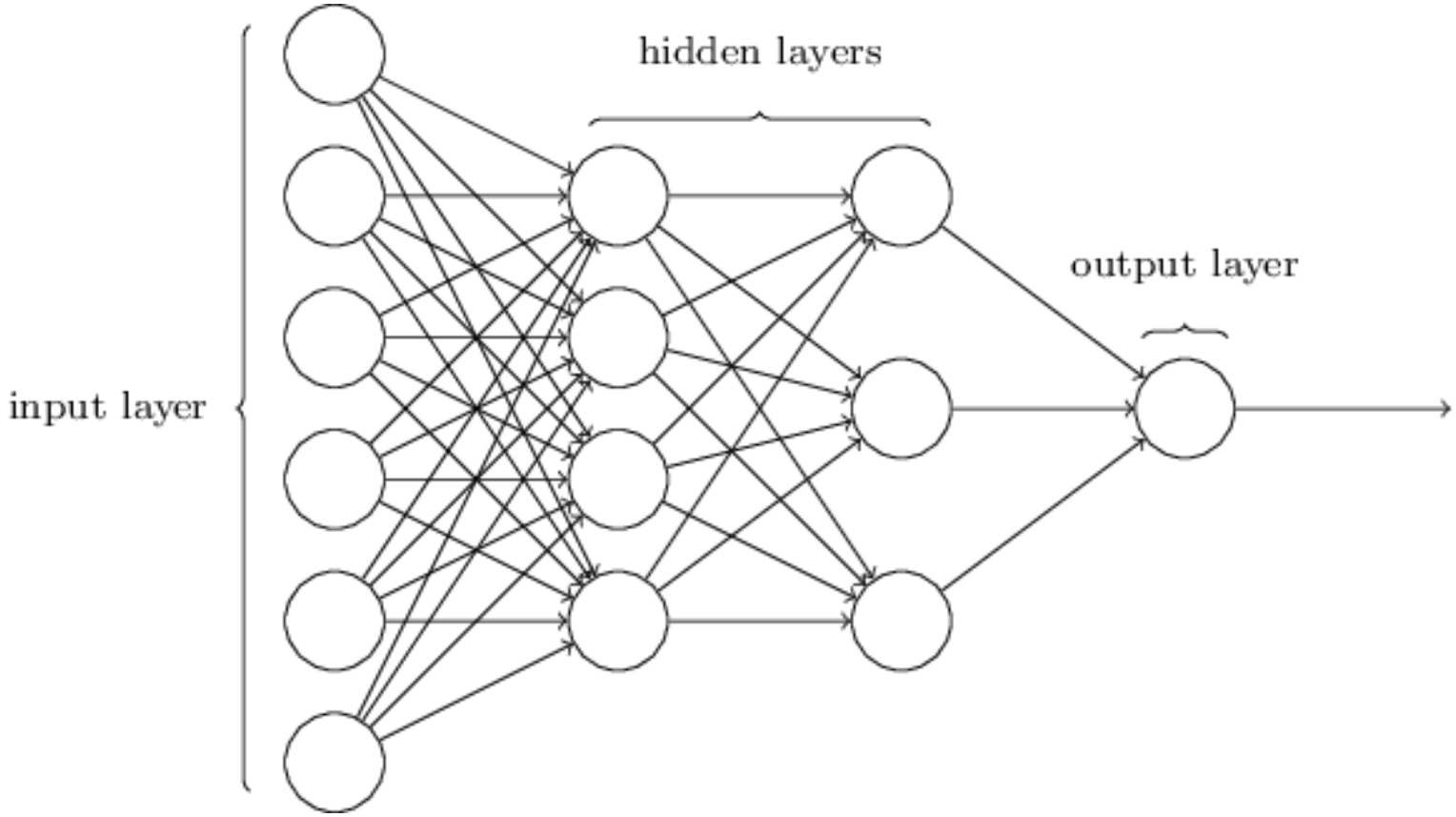


Image credit: https://www.simplilearn.com/deep-learning-tutorial

A Single Hidden Layer Network



A Deep Multilayer Network



Backpropagation

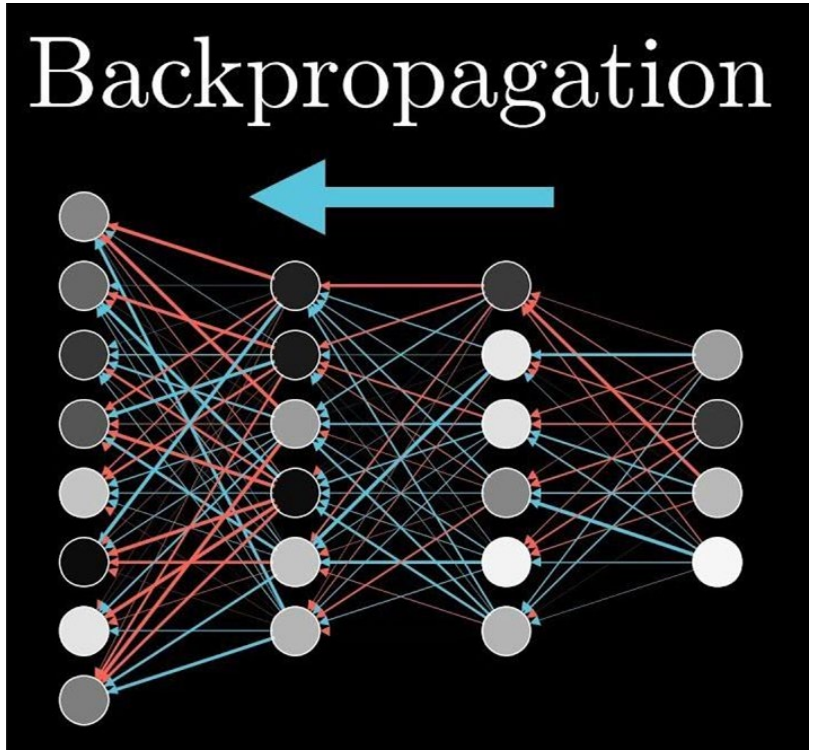
Backpropagation is the essence of neural net training

The practice of *fine-tuning the weights* of a neural net

Based on the error rate (i.e., loss) obtained in the previous epoch (i.e., iteration)

Proper tuning of the weights ensures lower error rate

Makes the model reliable by increasing its generalization



Reference:
https://www.youtube.com/watch?v=Ilg3gGewQ5U

Additional Information

Understanding how backpropagation trains:

https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/

Exploring deep neural networks (Tensor Flow playground):

Understanding neural networks with TensorFlow playground

Learning Resources

Free online tutorial, blogs, and videos available

- LinkedIn Learning at Syracuse University (for SU students, faculty and staff) (Login – Answers)
- *10 Free Courses for Machine Learning and Data Science* (https://www.kdnuggets.com/2018/11/10-free-must-see-courses-machine-learning-data-science.html)
- *R Markdown Notebooks for "Deep Learning with R"* (jjallaire/**deep-learning-with-r-notebooks**)

Deep Learning in R

School of Information Studies
Syracuse University

Deep Learning in R Using Keras

```
#Keras provides an R interface to the Python deep learning package Keras
install.packages('devtools’) devtools::install_github("rstudio/keras")

#Keras uses Tensorflow at backend to install
#Both Keras and Tensorflow run using the following commands:
library(keras)
install_keras()
```

Simple Example: Loading the Data

```
library(tidyverse)
library(keras)
library(caret)

df <- read_csv("testData.csv")
head(df, 5)
# A tibble: 5 x 4
     X1      X2      X3    y_data
  <dbl>   <dbl>   <dbl>    <dbl>
1 0.308   0.665   0.882        1
2 0.258   0.0175  0.318        0
3 0.552   0.409   0.454        0
4 0.0564  0.416   0.380        0
5 0.469   0.657   0.390        1
```

Can you figure out the “rule” for y_data?

Simple Example: Make Output Binary

```
#Change y into a matrix with a 0 or 1 for each choice
df$y_matrix <- to_categorical(df$Y, num_classes = 2)

df[1,]
# A tibble: 1 x 5
     X1     X2       X3       Y            y_matrix[,1]  [,2]
  <dbl> <dbl> <dbl>  <dbl>                    <dbl> <dbl>
1 0.308 0.665 0.882     1                        0       1

df[2,]
# A tibble: 1 x 5
     X1       X2       X3         Y      y_matrix[,1]  [,2]
  <dbl>  <dbl>    <dbl>  <dbl>                    <dbl> <dbl>
1 0.258  0.0175 0.318      0                        1     0
```

Simple Example: Create Test and Train

```
#Create test and train data sets
trainList <-
        createDataPartition(y=df$Y, p=.80, list=FALSE)

trainData <- df[trainList,]
testData <- df[-trainList,]
```

Simple Example: Define a Model

```
#units is the number of neurons
#ReLU (rectified linear unit) is linear (identity) for positive values, and zero for negative
#softmax is used in the final layer for categorical analysis.
#it maps the output to a [0,1] range in such a way that the total sum is 1, hence, is a probability distribution
#input_shape is the number of X inputs (we have 3 inputs)
model = keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu",  input_shape = 3) %>%
    layer_dense(units = ncol(df$y_matrix),  activation = "softmax")
```

Simple Example: Compile the Model

```
#loss == how to measure error
#Use “categorical_crossentropy” in classification problems where only one result can be correct
#optimizer == how to optimize during iterations
#optimizer_rmsprop is a good default
#metrics == list of metrics to be evaluated by the model
#Often use metrics = “accuracy”
compile(model,
        loss = "categorical_crossentropy",
        optimizer = optimizer_rmsprop(),
        metrics = "accuracy")
```

Simple Example: Run the Model

```
#epochs: the number of times that the learning algorithm will work through the entire training data set
#batch_size: the number of samples to work through before updating the internal model parameters;
#    64, 128, 256 are typical default numbers
#validation_split: the percentage to validate the model—avoid overfitting

#(the rest is to train the model)
x_data <- as.matrix(trainData[,1:3])
history = fit(model,
              x_data,
              trainData$y_matrix,
              epochs = 20,
              batch_size = 128,
```

Simple Example: Run the Model (cont.)

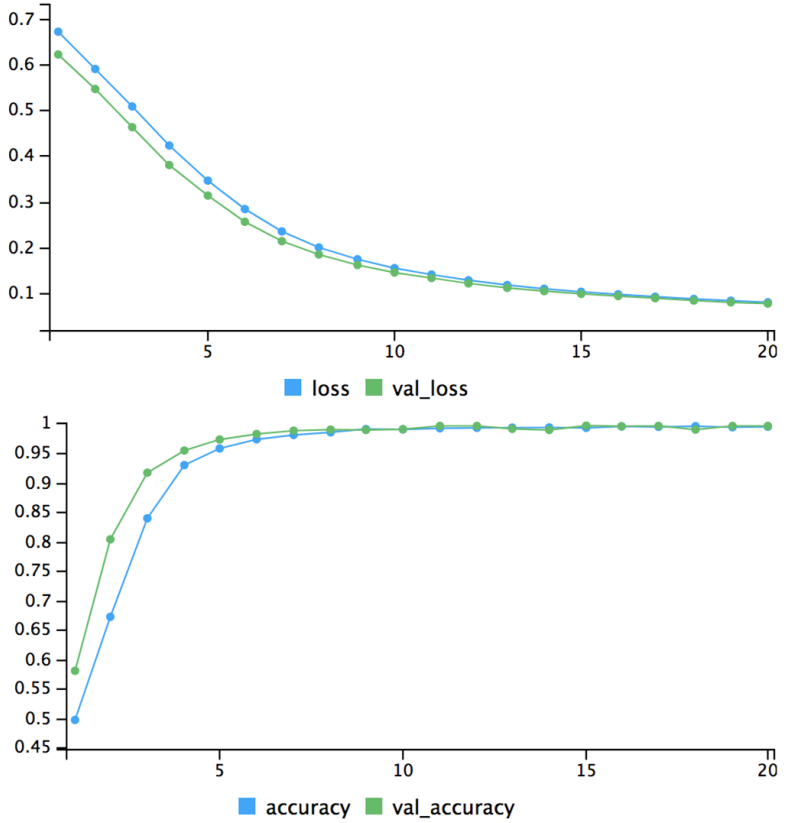
…

Epoch 19/20

loss: 0.0709 - accuracy: 0.9945 - val_loss: 0.0698 - val_accuracy: 0.9956

Epoch 20/20

loss: 0.0685 - accuracy: 0.9945 - val_loss: 0.0669 - val_accuracy: 0.9969



Simple Example: Use the Model

```
#Do prediction
x_data_test <- as.matrix(testData[,1:3])
y_data_pred=predict_classes(model, x_data_test)

#See how good the prediction was
confusionMatrix(as.factor(y_data_pred),  as.factor(testData$Y))
Confusion matrix and statistics
                    Reference
Prediction   0      1
         0           993     1
         1           10  997

               Accuracy : 0.995
```

Example 2: Use a Binary Model

```
#units is the number of neurons
#ReLU is linear (identity) for positive values, and zero for negative
#“sigmoid” is typically used in the final layer for binary analysis
#It maps the output to a [0,1] range
#input_shape is the number of X inputs (we have 3)
model = keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu",  input_shape = 3) %>%
    layer_dense(units =1,  activation = "sigmoid")
```

Example 2: Compile the Binary Model

```
#loss == how to measure error.
#Use binary_crossentropy in binary problems where one of two choices is correct
#optimizer == how to optimize during iterations
#optimizer_rmsprop is a good default
#metrics == list of metrics to be evaluated
#Often use metrics = “accuracy”
compile(model,
        loss = ”binary_crossentropy",
        optimizer = optimizer_rmsprop(),
        metrics = "accuracy")
```

Example 2: Run the Binary Model

```
#epochs: the number of times that the learning algorithm will work through the entire training dataset
#batch_size: the number of samples to work through before updating the internal model parameters;
#   64, 128, 256 are typical default numbers
#validation_split: the percentage to validate the model – avoid overfitting
#(the rest is to train the model)
x_data <- as.matrix(trainData[,1:3])
history = fit(model,
              x_data,
              trainData$Y,
              epochs = 20,
              batch_size = 128,
              validation_split = 0.2)
```

Simple Example 3: Use SVM

```
#Run the SVM
trainData$Y <- as.factor(trainData$Y)
svm.model <- train(Y ~ X1+X2+X3, data = trainData, method = "svmRadial",
      trControl=trainControl(method = "none"),  preProcess = c("center", "scale"))

#Use the model on the test data
testData$Y <- as.factor(testData$Y)
predictValues <- predict(svm.model, newdata=testData)
confusionMatrix(predictValues, testData$Y)
```

```
Confusion matrix and statistics
                    Reference
Prediction    0      1
             0  992      8
             1    11    989
       Accuracy : 0.9905
```

MIDST: A Slightly More Advanced Problem

Predict the digit in the image.

MNIST consists of 28 x 28 grayscale images of handwritten digits like below:

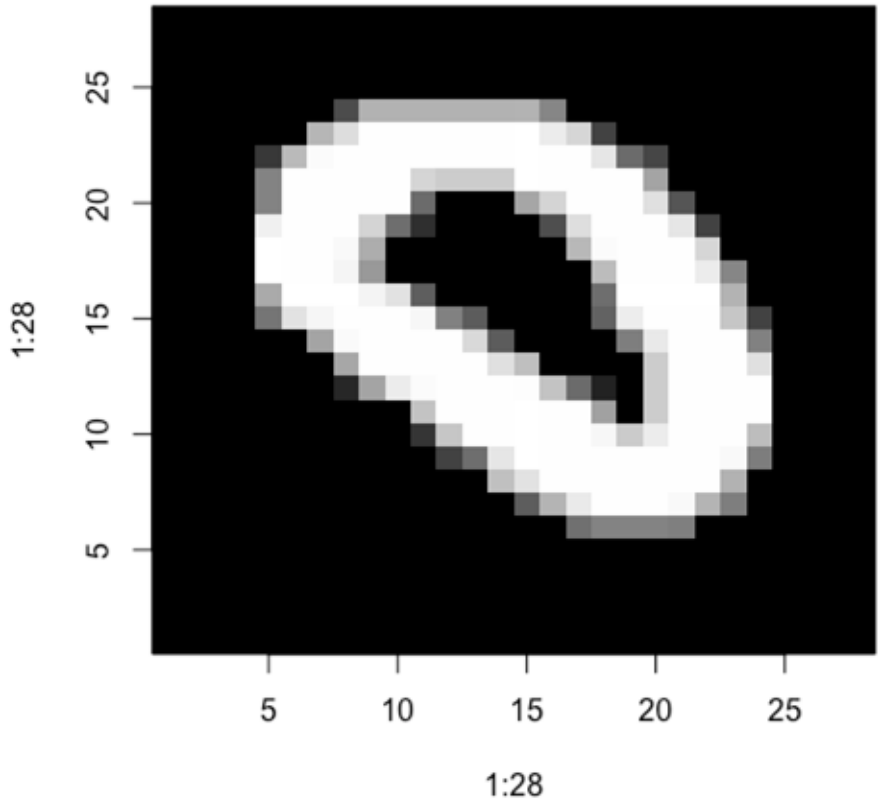
Reference:
Falbel D., Allaire JJ., Chollet F., RStudio, Google (n.d). Keras (n.d). Retrieved from https://keras.rstudio.com/

Loading MNIST Dataset

```
library(tidyverse)
library(keras)
library(caret)

#Load the images of digits from - https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
#Note: if loading a local file, need full path
mnist <- dataset_mnist(path = "/…/../mnist.npz")
```

Exploring the MNIST Dataset

```
#Look at an image (only 28x28) and the
# known value
img <- mnist$train$x[400,,]
image(1:28, 1:28, img, col =
gray.colors(start=0, end=1, n=256))

mnist$train$y[400]
[1] 0
```



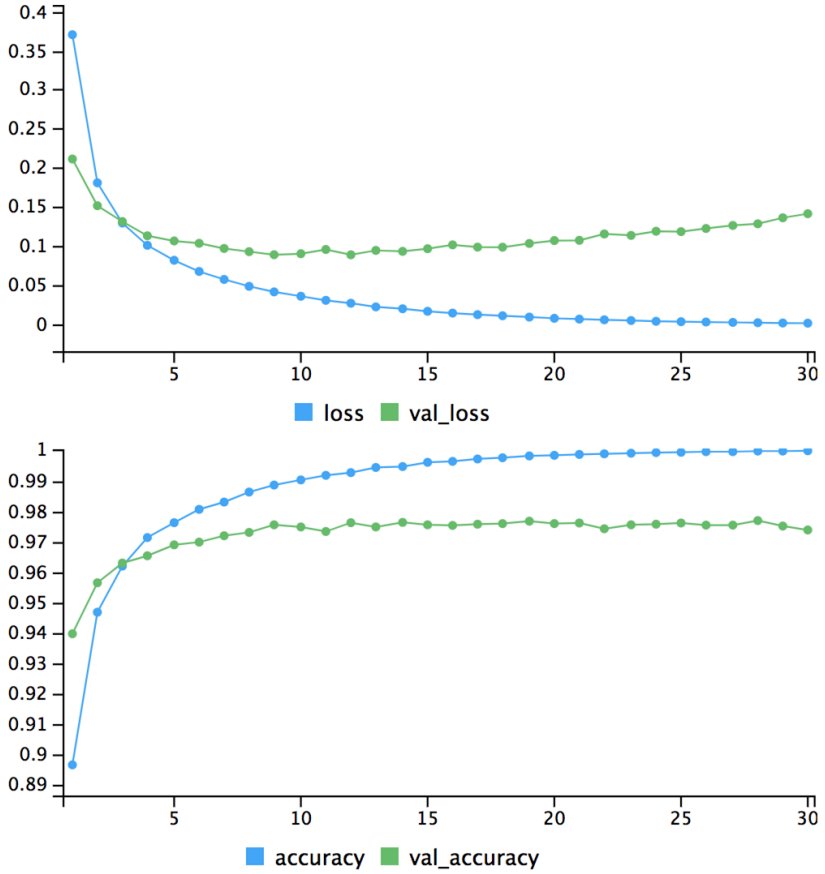
Define a MNIST Model

```
model <- keras_model_sequential() %>%
    #First "flatten" each 28x28 image into one vector of 784 numbers (28*28).
    #Think of this layer as unstacking rows of pixels in the image and lining them up—this layer
    only reformats the data.
    layer_flatten(input_shape=c(28, 28)) %>%

    #After the pixels are flattened, the network consists of two layers.
    #Start with 128 units (neurons)
    layer_dense(units = 128, activation = "relu") %>%

    #The second (last) layer is a 10-node softmax layer (since 10 possibilities).
    #The second layer returns an array of 10 probability scores that sum to 1.
    layer_dense(units = 10, activation = 'softmax')
```

Compile and Run the MNIST Model

```
#Same code as previous example to compile the model
model %>% compile(
     loss = 'categorical_crossentropy’,
     optimizer = optimizer_rmsprop(),
     metrics = c('accuracy')
)

#Same code as previous example to run the model
y_train <- to_categorical(mnist$train$y, 10)
history <- model %>% fit(
    mnist$train$x, y_train,
    epochs = 30, batch_size = 128,
    validation_split = 0.2
)
```

Explore the Output of the MNIST Model



Evaluating the Model

```
#Explore model accuracy
pred.digit <- model %>% predict_classes(mnist$test$x)
confusionMatrix(as.factor(pred.digit), as.factor(mnist$test$y))
```

Overall statistics

Accuracy : 0.9777

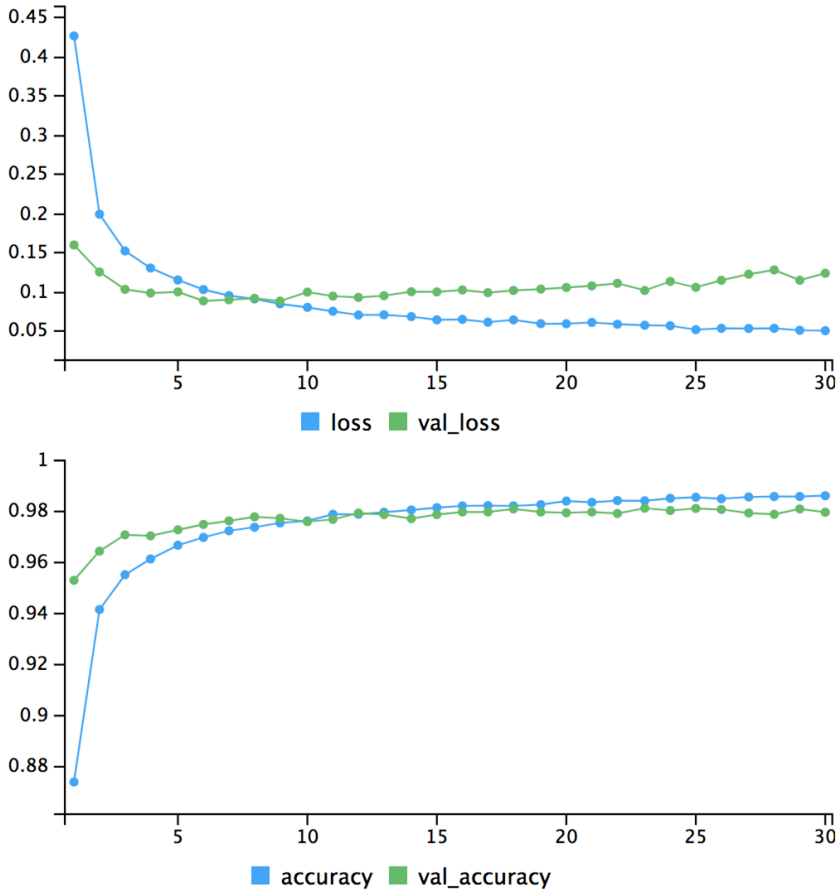
Defining a Better MNIST Model

```
#Now improve the model—add a third layer of nodes (neurons) and add
dropout layers
# (helps with overtraining, sets % of neuron weights to 0 randomly)
model <- keras_model_sequential() %>%
    layer_flatten(input_shape=c(28, 28)) %>%
    layer_dense(units = 256, activation = 'relu') %>%
    layer_dropout(rate = 0.4) %>%
    layer_dense(units = 128, activation = 'relu') %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 10, activation = 'softmax')
```

Compile and Run the MNIST Model

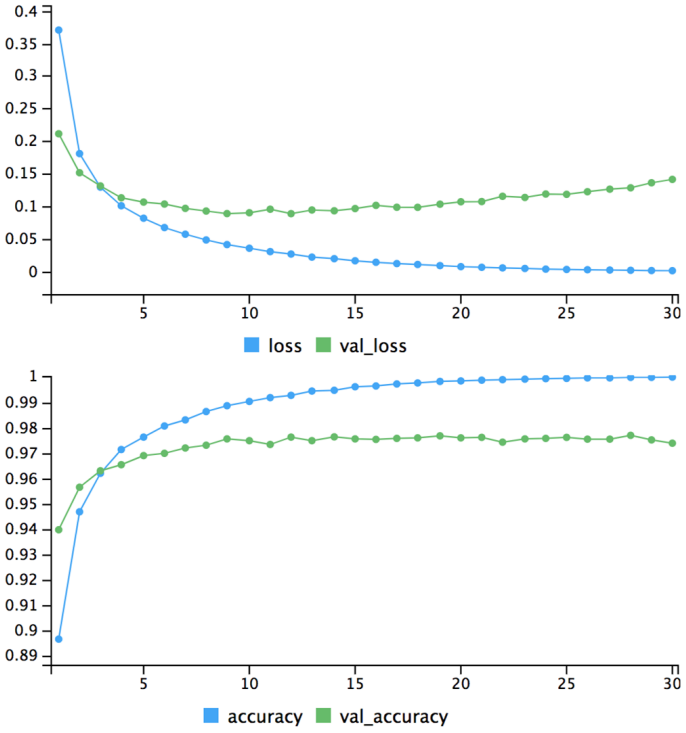
```
#Same code as previous example to compile the model
model %>% compile(
    loss = 'categorical_crossentropy',
    optimizer = optimizer_rmsprop(),
    metrics = c('accuracy')
)

#Same code as previous example to run the model
y_train <- to_categorical(mnist$train$y, 10)
history <- model %>% fit(
    mnist$train$x, y_train,
    epochs = 30, batch_size = 128,
    validation_split = 0.2
)
```

Explore the Output of the MNIST Model

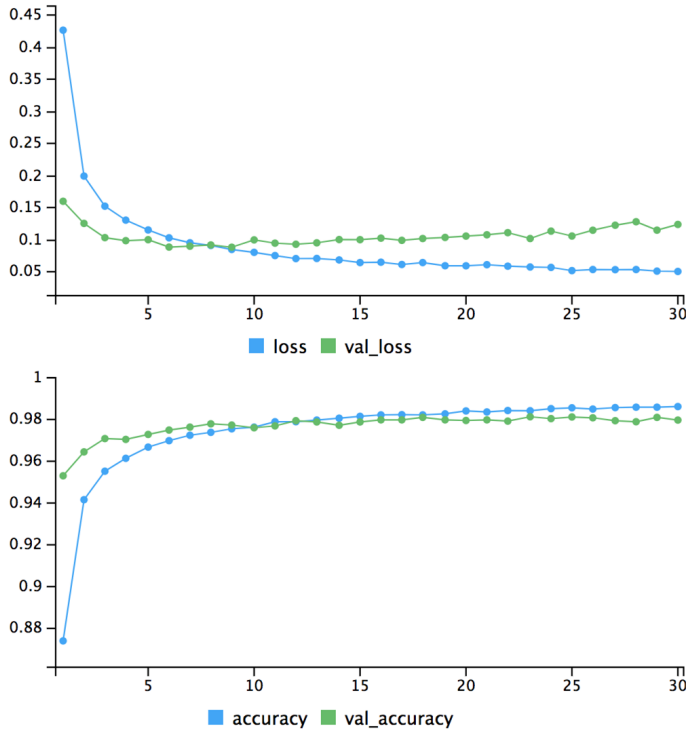


Explore the Output of the MNIST Model (cont.)

Initial model results

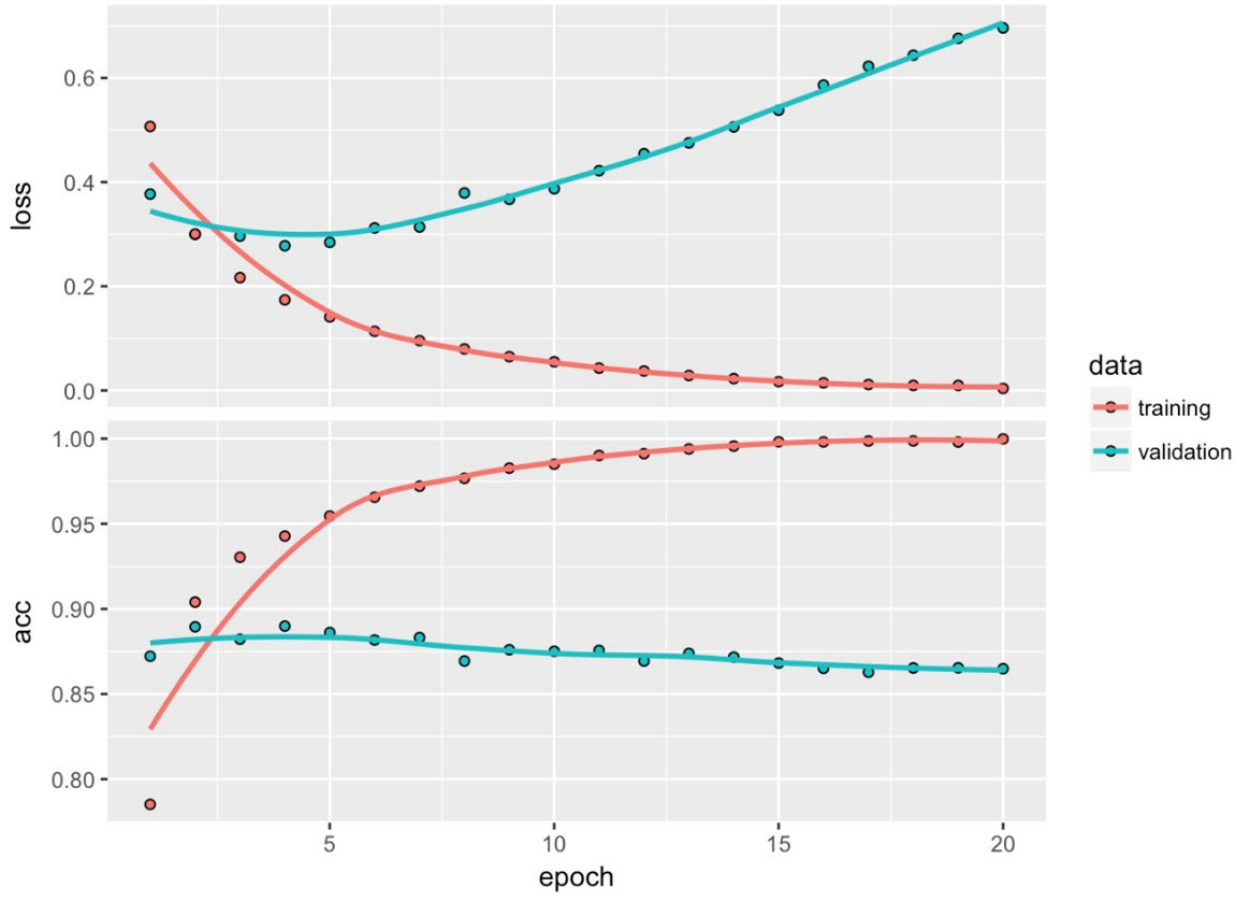


Updated model results



Note overfitting in the initial results

More Epochs Is Not Always Good!



Evaluating the Better Model

```
#Explore model accuracy
pred.digit <- model %>% predict_classes(mnist$test$x)
confusionMatrix(as.factor(pred.digit), as.factor(mnist$test$y))
```

Overall statistics

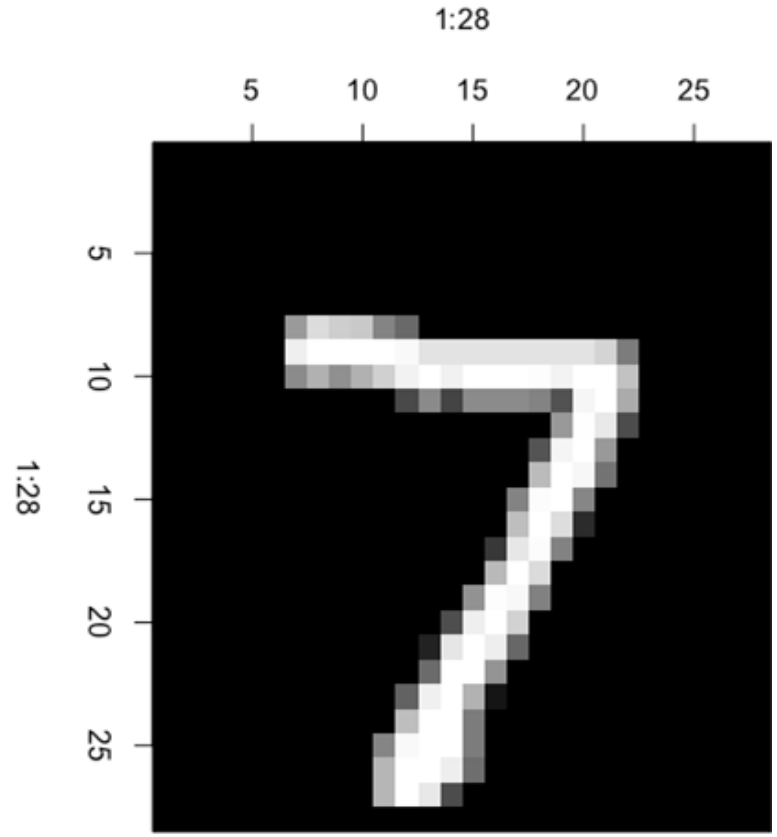
Accuracy: 0.9809

(previous accuracy was: 0.9777)

While not a significant better result for this “simple” example, this approach is used for more complicated data sets.

Using the Model

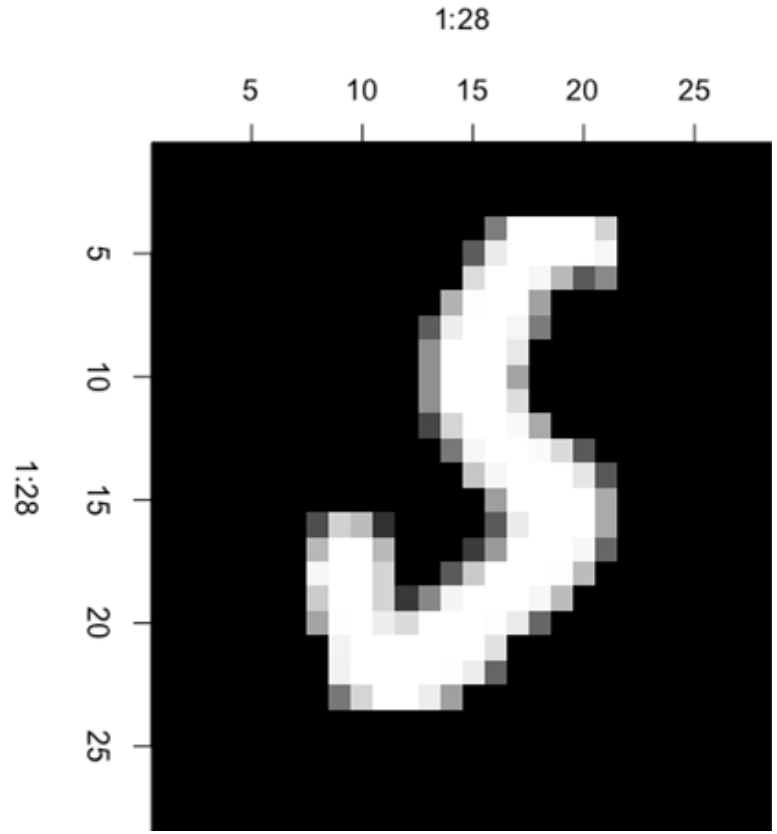
```
pred.digit[1]
[1] 7
mnist$test$y[1]
[1] 7
```




```
img <- mnist$test$x[1,,]
image(1:28, 1:28, img, col =
gray.colors(start=0, end=1, n=256))
```

Where Did the Model Mess Up?

```
which(pred.digit != mnist$test$y)
  [1]  248  322  341  446  …. 9983

pred.digit[9983]
[1] 6

mnist$test$y[9983]
[1] 5

img <- mnist$test$x[9983,,]
image(1:28, 1:28, img, col = gray.colors(start=0, end=1,
n=256))
```



Question

Could you do this digit recognition using SVM?

Deep Learning in R (cont.)

School of Information Studies
Syracuse University

Could Use an SVM - but Takes **Much** Longer

```
#Try SVM
library(caret);  library(e1071)
set.seed(1000)

#Test and train data
x_train <- array_reshape(mnist$train$x, c(nrow(mnist$train$x), 784))
x_test <- array_reshape(mnist$test$x, c(nrow(mnist$test$x), 784))
trainData <- as.data.frame(x_train)
trainData <- cbind(trainData, digit=mnist$train$y)

#Run the SVM—takes about 20–90 minutes on a high-end laptop
trainData$digit <- as.factor(trainData$digit)
svm.model <- train(digit ~ ., data = trainData,  method = "svmRadial",
                   trControl=trainControl(method = "none"), preProcess = c("center", "scale"))

#Actual resulting model is good (close to previous model), but clearly not scalable
```

Saving/Loading a Model

```
#After training completes, you can save your model
model %>% save_model_hdf5("my_model.h5")

#Call load_model_hdf5 to load the model
new_model <- load_model_hdf5("my_model.h5")

#Viewing the loaded model summary
new_model %>% summary()
```

ImageNet

Contains millions of images with thousands of categories

Training for this would need a lot of resources and time (requires GPUs)

But prebuilt models are available to use

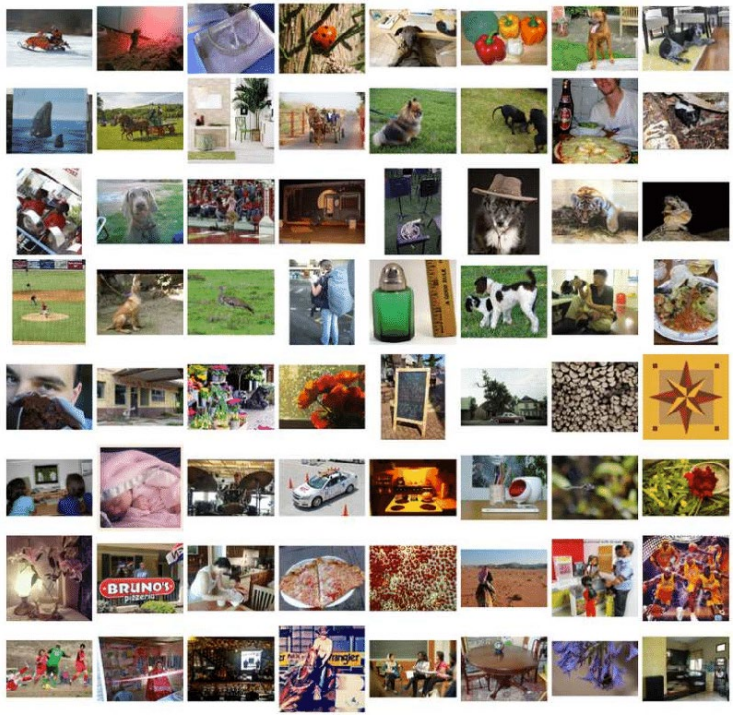


Image credit:
https://www.researchgate.net/figure/Examples-in-the-ImageNet-dataset_fig7_314646236

Loading a Pretrained Model

Xception: an ImageNet model

- Trained on 1,000 classes from ImageNet
- Trained using 60 NVIDIA K80 GPUs
- Training took 3 days!

```
#Call load_model_hdf5 to load the model if you have model file
model <- load_model_hdf5("xception.h5")

#Alternatively, one could download and load the trained model using
#“application_xception” function from keras
model <- application_xception(include_top = TRUE,
                              weights = "imagenet", input_tensor = NULL,
                               input_shape = NULL, pooling = NULL, classes = 1000)
```

Reference:
Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1251-1258).
https://cran.rstudio.com/web/packages/keras/vignettes/applications.html

Get Ready to Use the Model

```
#Compile the model
model %>% compile(
    loss = 'categorical_crossentropy',
    optimizer = optimizer_rmsprop(),
    metrics = c('accuracy')
)

#Be able to decode predictions to animal names
imageNames <- fromJSON("imagenet_class_index.json")
imageNamesDF <- as.data.frame(imageNames,   stringsAsFactors = FALSE)
predictedImageNames <- as.character(imageNamesDF[2,1:1000])

predictedImageNames[1:3]
[1] "tench"           "goldfish"         "great_white_shark"
```

Use the Pretrained Model (See if the Model Can Identify a Cat)

```
#Use the model on a test image (of a cat)
testImage <- "cat.jpg"

#First just display image using magick package
library(magick)
im <- image_read(imageName)
plot(im)
```

Image from: https://pixabay.com/photos/cat-small-kitten-domestic-cat-pet-4611189/

Use Model to Make a Prediction

```
#Next load image, convert to array and reshape for our model
img <- image_load(testImage, target_size = c(299,299))
x <- image_to_array(img)
x <- array_reshape(x, c(1, dim(x)))

#xception_preprocess_input() should be used for keras image preprocessing
x <- xception_preprocess_input(x)

#Predict the type of image
predictions <- model %>% predict(x)

#Create a dataframe of prediction percentage and the name of that animal
predsDF <-   data.frame(predPercent=predictions, predictedImageNames)
```

What Did the Model Predict?

```
#Sort and show the predictions
predsDF <- predsDF %>% arrange(desc(predPercent))
predsDF[1:5,]
```

	predPercent	predictedImageNames
1	0.555413783	tabby
2	0.215213165	tiger_cat
3	0.047023088	Egyptian_cat
4	0.031442393	lynx
5	0.002431044	sock

Use the Pretrained Model (See if the Model Can Identify a Ball)

```
#Use the model on a test image (of a ball)
testImage <- ”soccerBall.jpg"

#First just display image using magick package
library(magick)
im <- image_read(imageName)
plot(im)
```

Image from: https://pixabay.com/photos/football-duel-rush-ball-sport-1331838/

Use Model to Make a Prediction

```
#Next load image, convert to array and reshape for our model
img <- image_load(testImage, target_size = c(299,299))
x <- image_to_array(img)
x <- array_reshape(x, c(1, dim(x)))

#xception_preprocess_input() should be used for keras image preprocessing
x <- xception_preprocess_input(x)

#Predict the type of image
predictions <- model %>% predict(x)

#Create a dataframe of prediction % and the name of that animal
predsDF <-   data.frame(predPercent=predictions,predictedImageNames)
```

What Did the Model Predict?

```
#Sort and show the predictions
predsDF <- predsDF %>% arrange(desc(predPercent))
predsDF[1:5,]
```

	predPercent	predictedImageNames
1	0.8440486789	soccer_ball
2	0.0065183444	rugby_ball
3	0.0035442952	football_helmet
4	0.0010038349	baseball
5	0.0009614553	croquet_ball

Building a Shiny App

School of Information Studies
Syracuse University

Shiny Apps Have Two Pieces

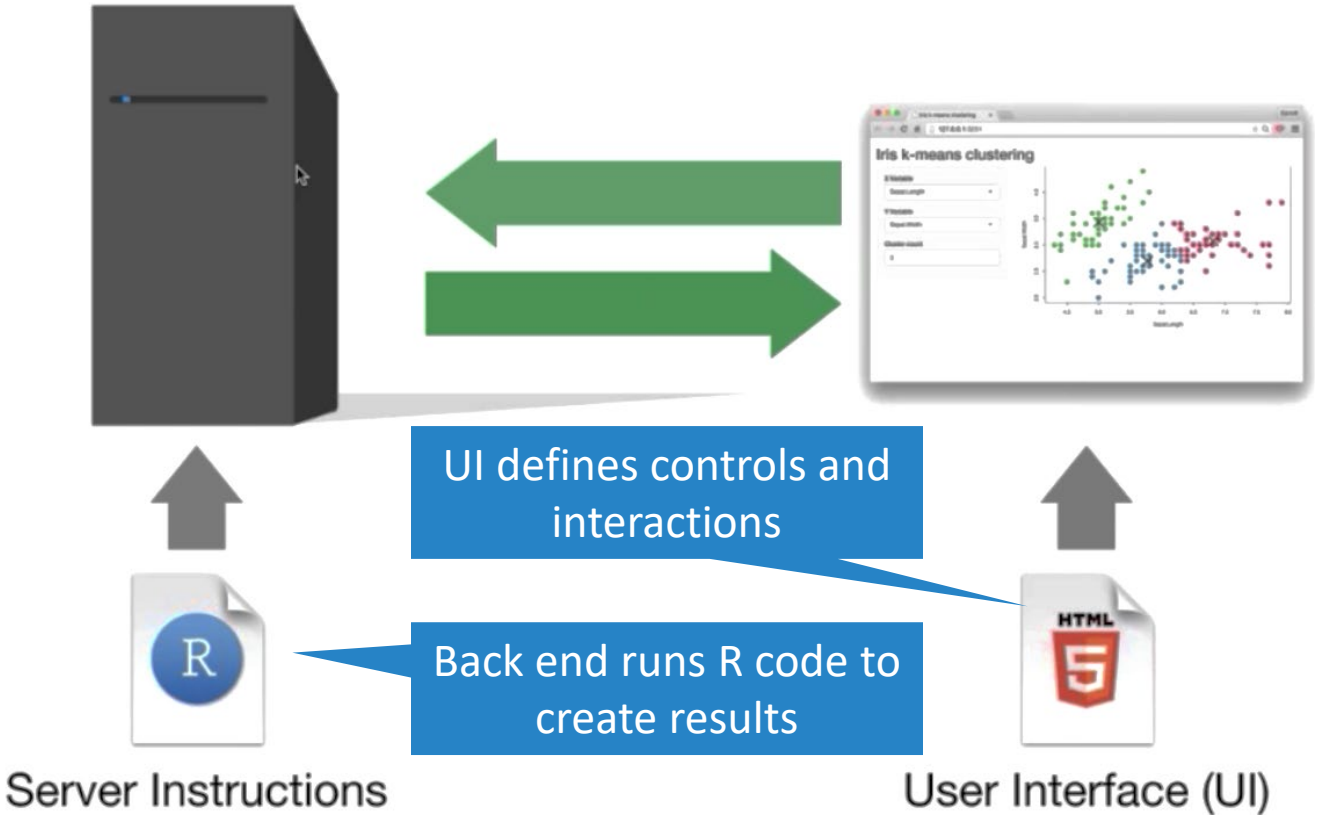
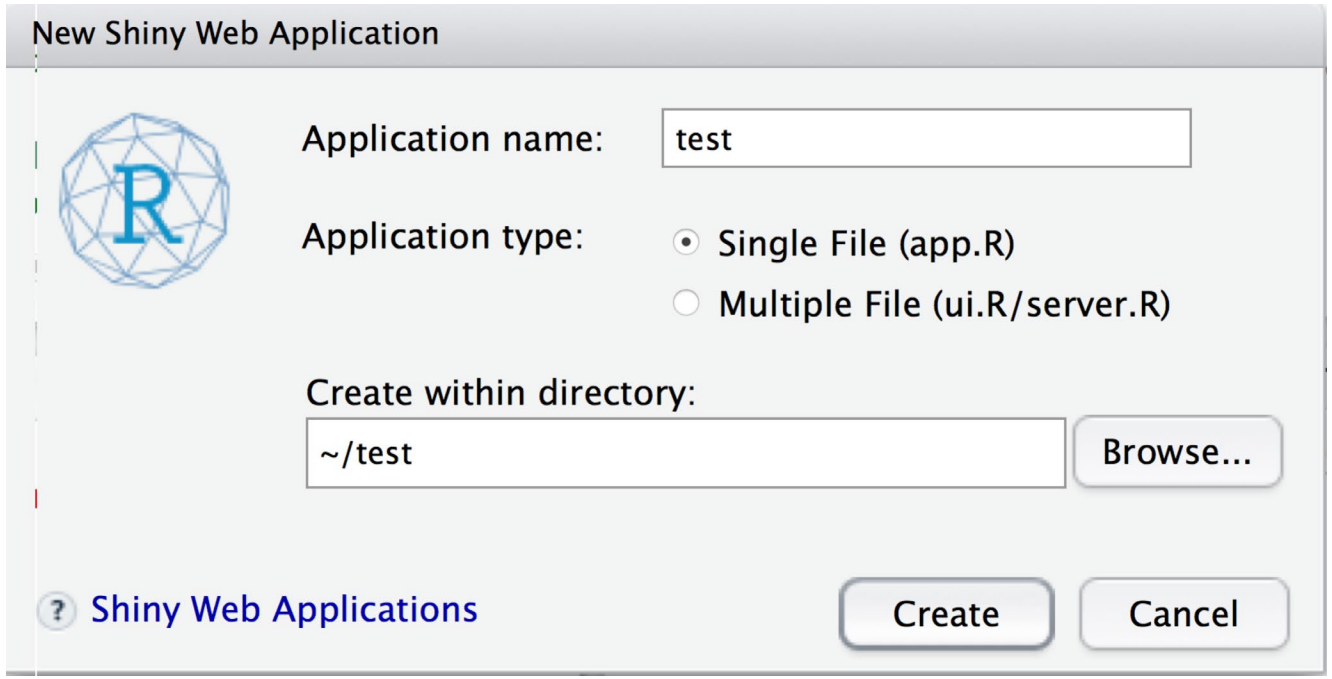


Image credit: R-Studio

A New Type of File

A "Shiny Web App…"

Not an R Script



UI Code

```
#Define UI for application that draws a histogram
ui <- fluidPage(

  #Application title
  titlePanel("Old Faithful Geyser Data"),

  #Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins", "Number of bins:", min = 1,  max = 50, value = 30
    ),

    #Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot") ) ) )
```

This translates inside the server to a mess of HTML5 code that implements the visible UI on the web page.

Here's the connection to some R output; see next slide.

Server Code

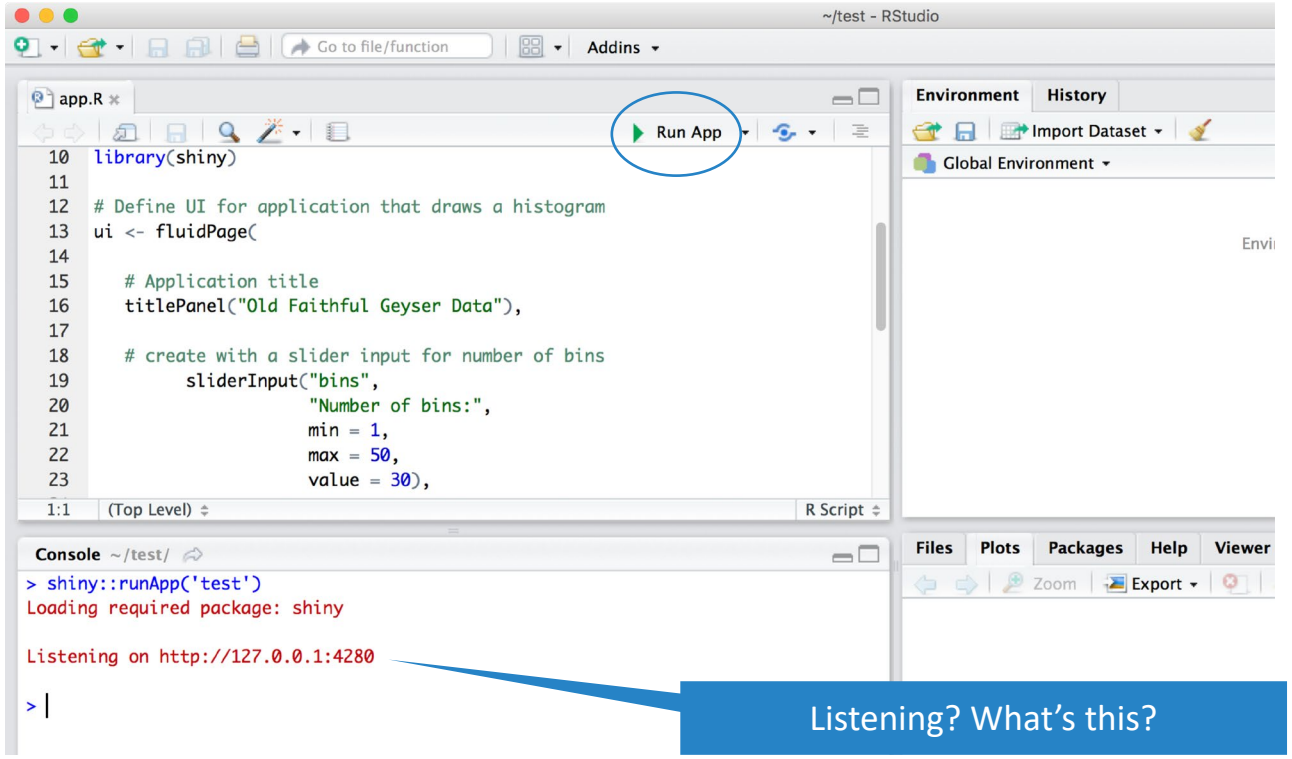
```
server <- function(input, output) {
  output$distPlot <- renderPlot({
    #Generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out =  input$bins + 1)

    #Draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
}
```

There's the other side of the connection to the UI.

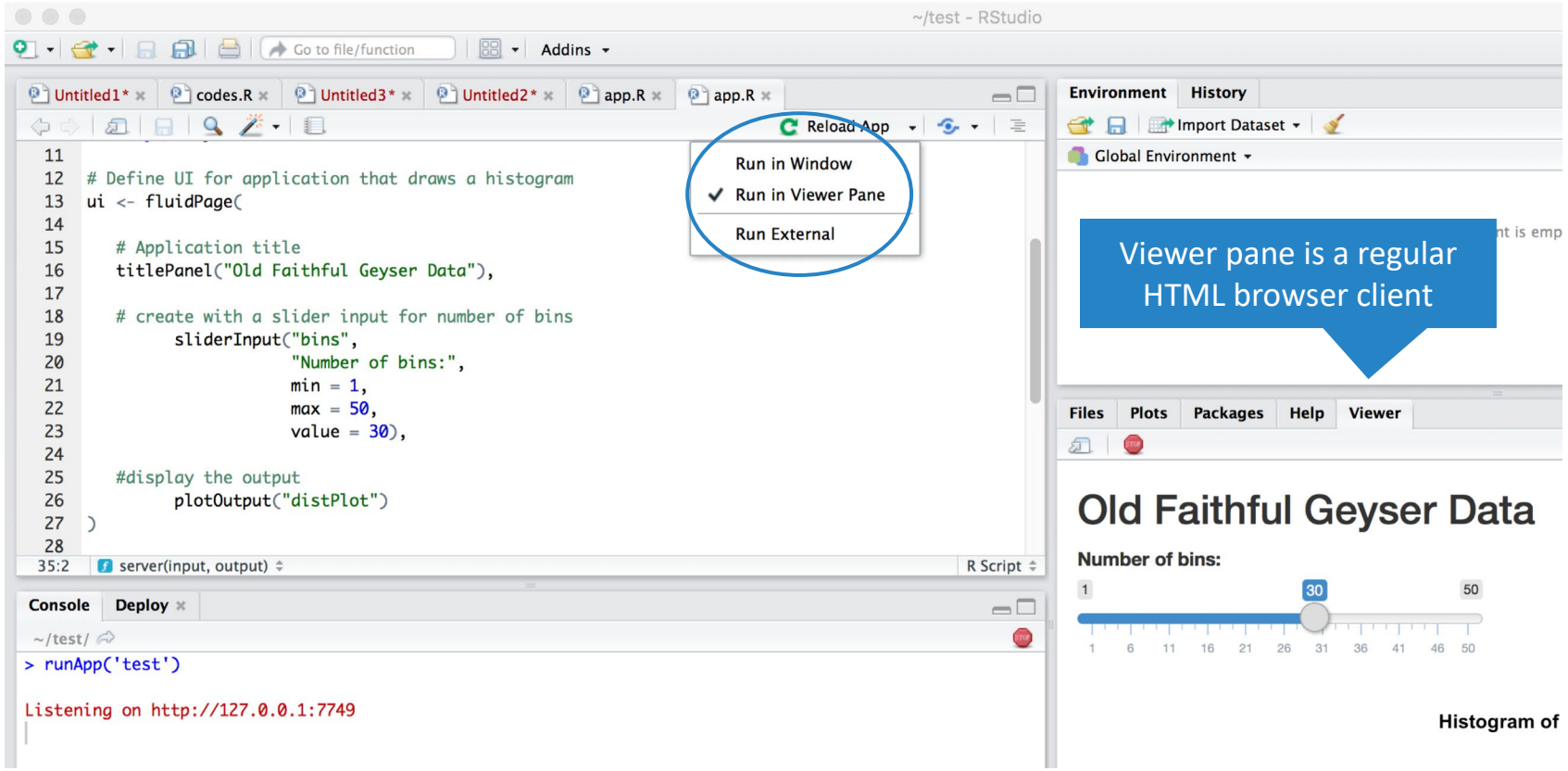
Plain vanilla R code: The resulting plot output gets rendered into HTML5 and delivered to the server that is providing the HTTPS response to your browser.

Running the Application

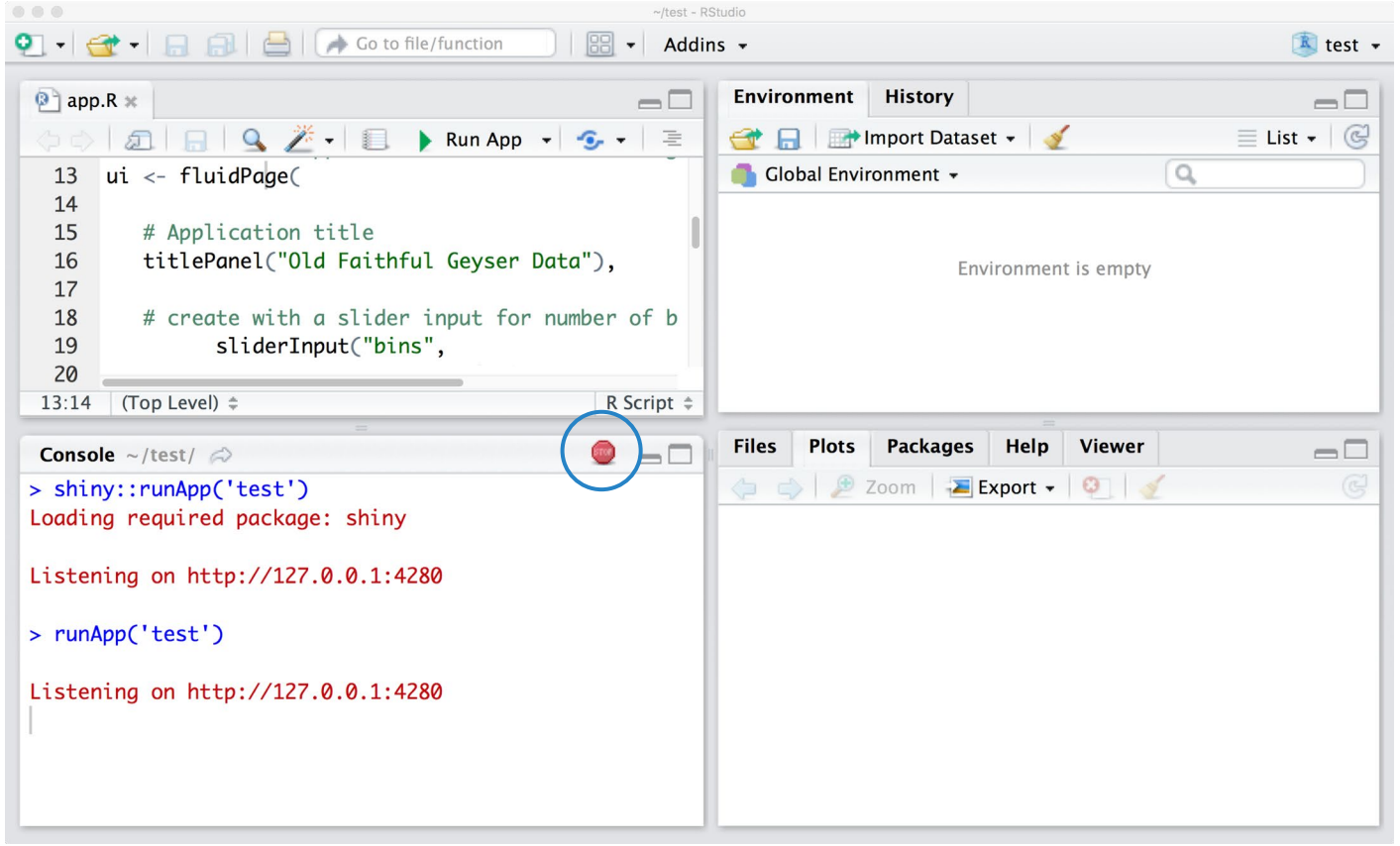
shinyApp(ui = ui, server = server)



Many Ways to Run the App



How to Stop the App

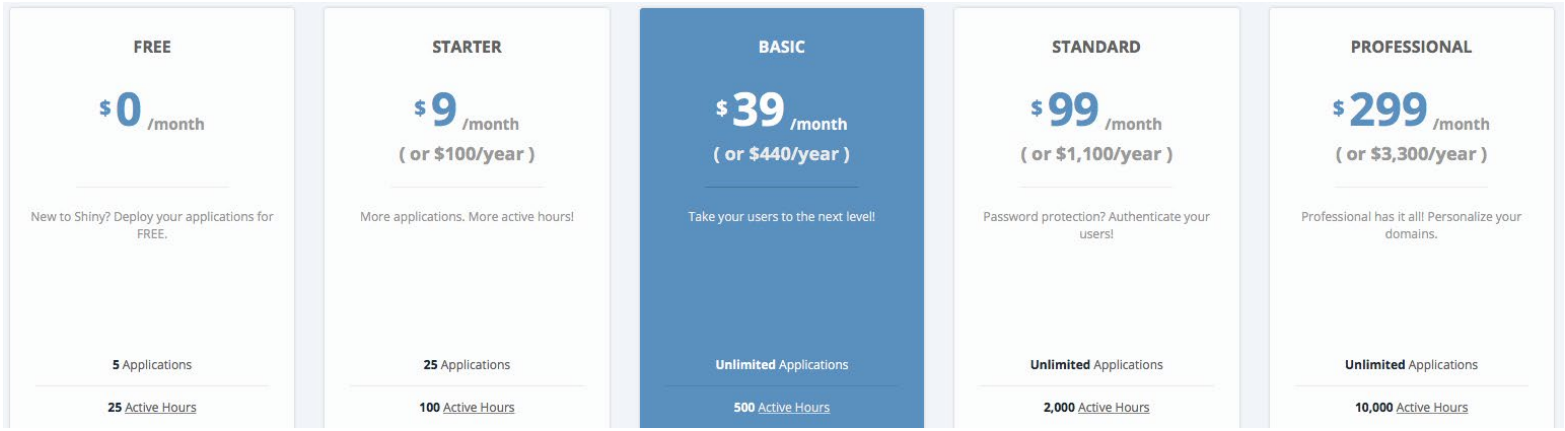


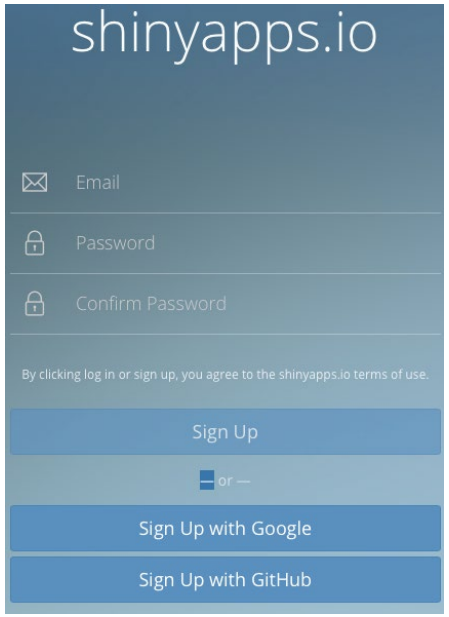
Shinyapps.io

Sign up for a free account

You will get a token and a secret

These will be used with the rsconnect package to communicate with your account





Deploying the Application

Use Shinyapps,

```
> install.packages("rsconnect")
> library(rsconnect)
```

```
> setAccountInfo(name='xxx',
+   token='yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy',
+   secret=zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz')
```

Need a Shiny app account

Publishing the Application