SMU
SINGAPORE MANAGEMENT UNIVERSITY

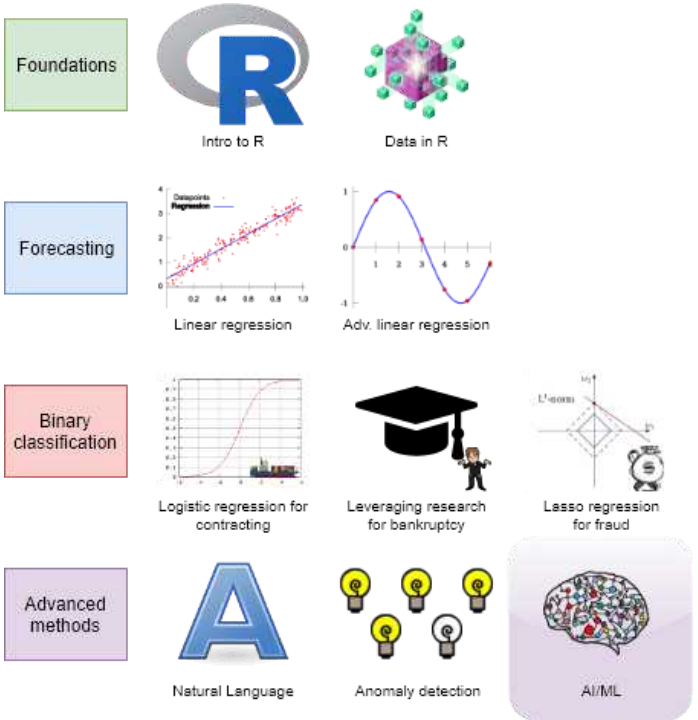
Forecasting and Forensic Analytics

Session 11: Neural Networks and Deep Learning

Dr. Wang Jiwei

Preface

Learning objectives



- **Theory:**
  - Neural Networks
  - Deep Learning
- **Application:**
  - Image recognition
  - Credit card fraud detection
- **Methodology:**
  - Keras
  - Tensoflow
  - Neural networks

Reminder: Group project and presentation

1. Keep moving moving moving .....
2. Up to 20 submissions to Kaggle.com per day (to check your out of sample accuracy)
3. For project submission and presentation:
   - Deadline: by this Sunday 1159pm
   - Submission: (1) report and code (RMarkdown preferred), (2) data, (3) presentation deck
     - zip all into one file and submit to eLearn
     - if too large (>50M), please share with me through OneDrive
     - I should be able to run your code from the submission files directly, ie, keep all your data and folder structure
   - Groups 11 & 12: video submission within 3 days after the last session. all must present, face must be seen on video, share the video with me directly.

Languages for ML/AI

R/Python for ML/AI

R Packages we have covered

- `package:glmnet`: LASSO and elastic nets
- `package:xgboost`: XGBoost
- `package:caret`
- `package:keras`: Plugs into python's Keras
- `package:H2O`: Plugs into python's H2O

Other useful packages

- `package:Prophet`: ML for time series forecasting
- `package:H2O4GPU`: Plugs into python's H2O4GPU
- `package:spacyr`: Plugs into python's SpaCy

Python platform for this course:

- **TensorFlow** (Google)
  - Can do everything
- H2O
- TPOT

Other notable ones:

- pytorch -- facebook, biggest threat to TensorFlow
- gensim: "Topic modelling for humans"
- caffe (Berkley)
- caffe2 (Facebook)
- SpaCy -- Fast NLP processing
- CoreNLP -- through various wrappers to the Java library
- Scikit learn -- one stop shop for most older libraries

TensorFlow is the leader

- It has strong support from Google and others
  - TensorFlow Hub -- Premade algos for text, image, and video
  - tensorflow/models -- Premade code examples
    - The research folder has amazing resources
  - tensorflow/tensor2tensor -- AI research models
- It can run almost ANY ML/AI/NN algorithm
- It has APIs for easier access
- It can deploy anywhere
  - Python & C/C++ built in
  - TensorFlow light for mobile deployment
  - TensorFlow.js for web deployment

Neural Networks

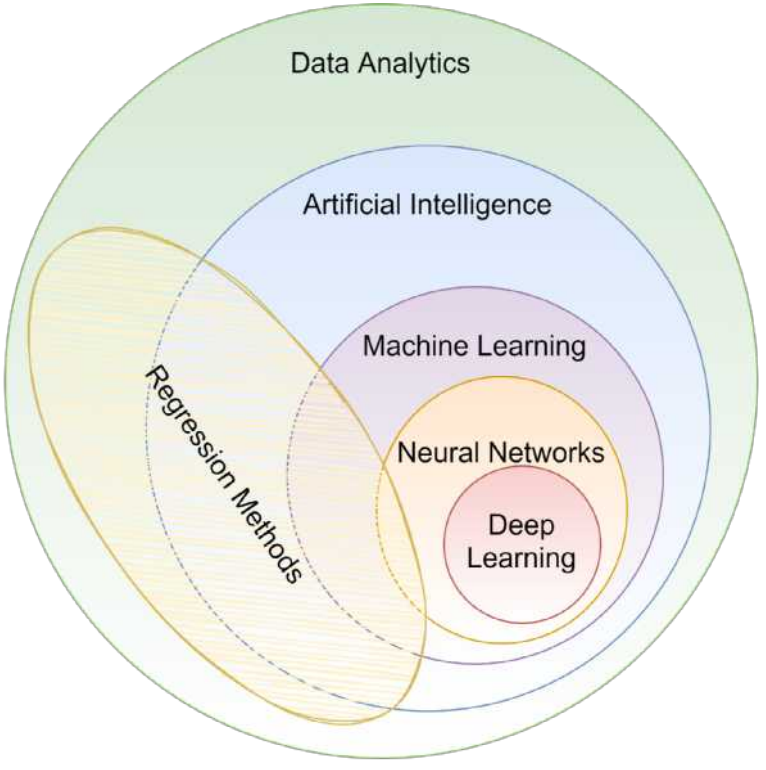
Turing Award 2018

- ‘Godfathers of AI’ honored with Turing Award, the Nobel Prize of computing
  - Bengio: University of Montreal;
  - Hinton: University of Toronto, Google;
  - LeCun: New York University, Facebook.

> "developed conceptual foundations for the field, identified surprising phenomena through experiments, and contributed engineering advances that demonstrated the practical advantages of deep neural networks"

What are neural networks?

- The phrase *neural network* is thrown around almost like a buzz word
- *Neural networks* are actually a specific type class algorithms
  - There are many implementations with different primary uses

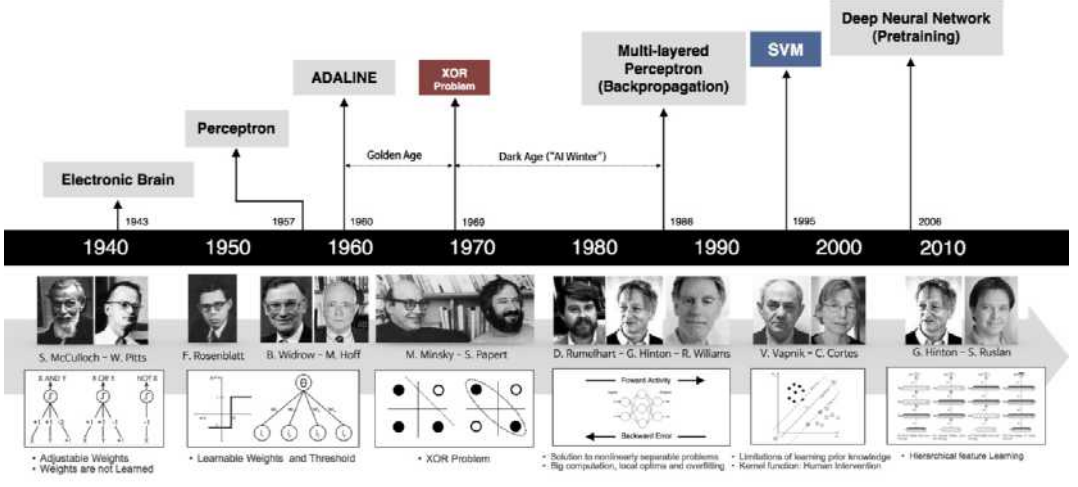


What are neural networks?

- Originally, the goal was to construct an algorithm that behaves like a human brain (neuron)
  - "neuron"" is the basic working unit of our brain
- Current methods don't quite reflect human brains, however:
  1. We don't fully understand how our brains work, which makes replication rather difficult
  2. Most neural networks are constructed for specialized tasks (not general tasks)
  3. Some (but not all) neural networks use tools our brain may not have
     - I.e., backpropogation is potentially possible in brains, but it is not pinned down how such a function occurs (if it does occur)

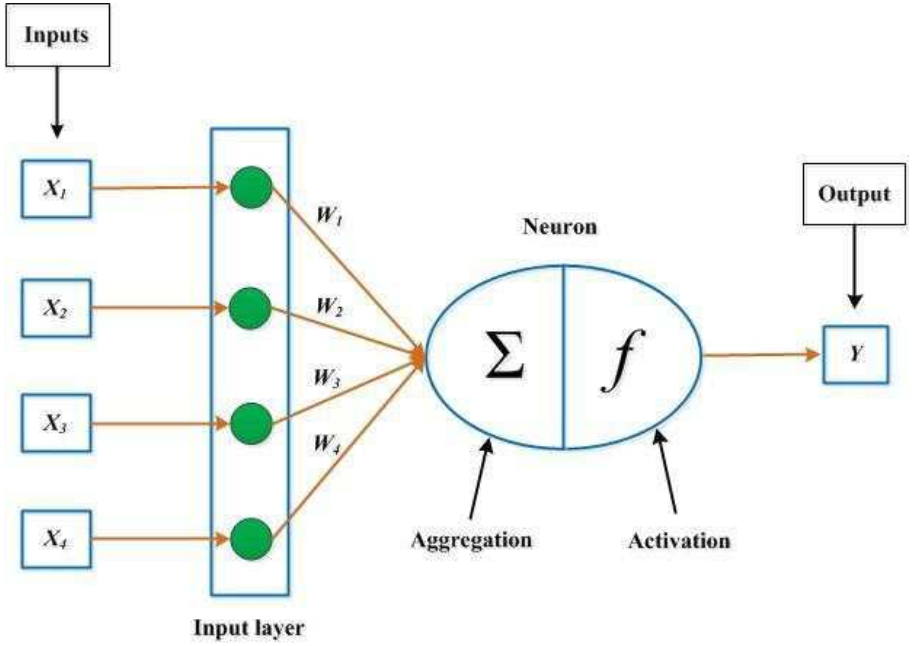
What are neural networks?

- Neural networks are a method by which a computer can learn from observational data
- In practice:
  - They were not computationally worthwhile until the mid 2000s
  - They have been known since the 1950s (perceptrons)
  - They can be used to construct algorithms that, at times, perform better than humans themselves
    - But these algorithms are often quite computationally intense, complex, and difficult to understand
  - Much work has been and is being done to make them more accessible



Single artificial neuron

- The diagram defines a relationship between the *input signals* (x variables) and the *output signal* (y variables)
- Each input signal is *weighted* (w values) according to its importance.
- The input signals are *summed* by the cell body and the signal is passed on according to an *activation function (f)*

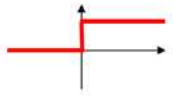
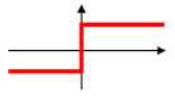
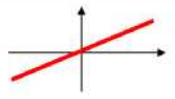
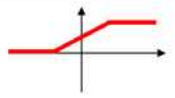
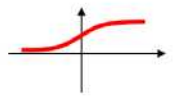
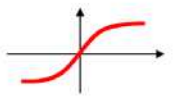
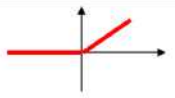
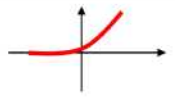


Building blocks of NN

- An `activation function`, which transforms a neuron's combined input signals into a single output signal to be broadcasted further in the network
- A `network topology` (or architecture), which describes the number of neurons in the model as well as the number of layers and manner in which they are connected.
- The `training algorithm` that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal

> variations of the building blocks will be used to construct various neural networks

Activation function

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

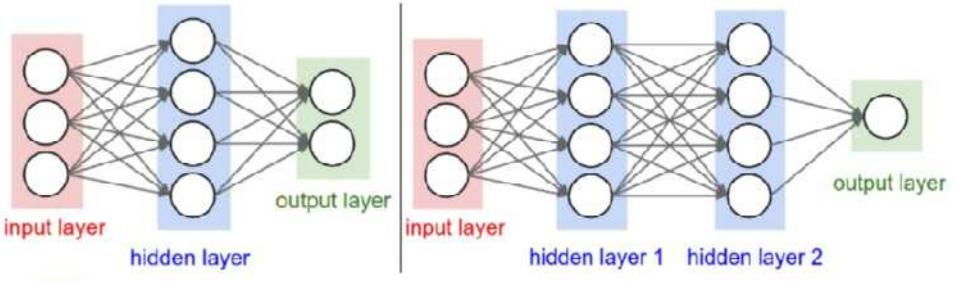
Copyright © Sebastian Raschka 2016 (http://sebastianraschka.com)

Squasing functions

- For most activation functions, the range of input values that affect the output signals is relatively narrow.
  - eg, for sigmoid, the output signal is always nearly 0 or 1 for an input signal below -5 or above +5, repectively
- The compression of signal in this way results in a saturated signal at the high and low ends of very dynamic inputs.
- Because this essentially squeezes the input values into a smaller range of outputs, activation functions like the sigmoid are sometimies called `squashing functions`.
- Solution: transform all inputs within a small range around 0, ie, `standardizing or normalizing` the features.

Network topology

- Network topology has three important characteristics:
  - the number of layers (`deep learning` has multple layers)
  - signal to travel backward or not (`feedforward` is one direction; `feedback` which allows in both directions, ie, with `short-term memory` or `delay`)
  - the number of nodes within each layer

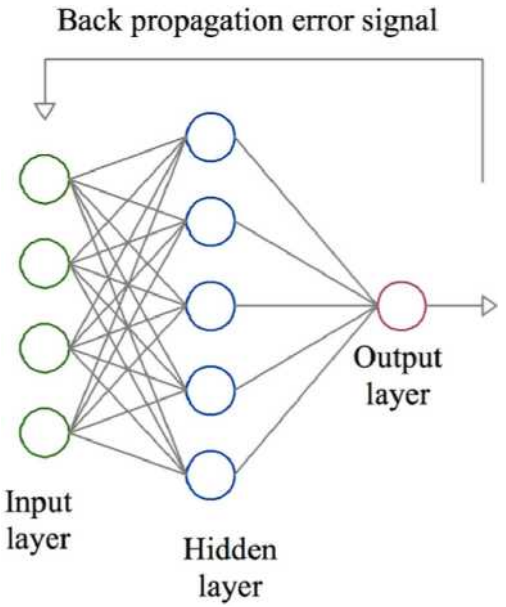


Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs. Right: A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

Naming conventions. Notice that when we say N-layer neural network, we do not count the input layer. Therefore, a single-layer neural network describes a network with no hidden layers (input directly mapped to output). In that sense, you can sometimes hear people say that logistic regression or SVMs are simply a special case of single-layer Neural Networks. You may also hear these networks interchangeably referred to as *"Artificial Neural Networks"* (ANN) or *"Multi-Layer Perceptrons"* (MLP). Many people do not like the analogies between Neural Networks and real brains and prefer to refer to neurons as *units*.

Training algo: backpropagation

- The backpropagation algorithm iterates through many cycles of two processes:
  - A `forward phase`: the neurons are acivated in sequence from the input layer to the output layer, and an output signal is produced
  - A `backward phase`: the output signal from the forward phase compared with the actual output in the training data. The difference (error) is propagated backwards to modify the weights and reduce future errors



Types of neural networks

- There are *a lot* of neural network types
  - See The "Neural Network Zoo"
- Some of the interesting ones:
  - RNN: Recurrent Neural Network
  - LSTM: Long/Short Term Memory
  - CNN: Convolutional Neural Network
  - DAN: Deep Averaging Network
  - GAN: Generative Adversarial Network
  - VAE (Variational Autoencoder): Generating *new* data from datasets

Deep learning with TensorFlow

Keras + TensorFlow

- `TensorFlow`: an open-source platform for ML/DL, written in C++, Python and CUDA (for GPU) by Google
- `Keras`: API written in Python, running on top of TensorFlow.
- We will use the R interface to Keras/TensorFlow

Install TF and Keras on Python

- Check your Python installation
  - Check by typing "python --version" in a command prompt
  - No Python? Install through Anaconda (with Python 3.8)
  - Python 2.7? TensorFlow needs support of Python 3.6 - 3.8.
  - Don't install the Python 3.9
- Choose one method to install TensorFlow and Keras on Python
  - If option 1 not working, try option 2

Install TensorFlow through RStudio

```
install.packages("tensorflow")
library(tensorflow)
install_tensorflow()
```

Install TensorFlow through Conda

- (1) run Anaconda Powershell Prompt as Administrator
- (2) `conda update conda`
- (3) `conda update anaconda`
- (4) `conda install tensorflow`
- (5) `conda install keras`

Install R API to Keras/TensoFlow

- Install with: `devtools::install_github("rstudio/keras")`
- Then finish the install in one of two ways:

CPU Based, works on *any* computer

- install this version first even if you want to use GPU version (GPU is not needed for this course)

```
library(keras)
install_keras()
```

Nvidia GPU based (For Windows and Linux only)

- Install the Software requirements first

```
library(keras)
install_keras(tensorflow = "gpu")
```

- If you are using Anaconda for Python, you may experience failure of Anaconda Prompt after installation of Keras and TensorFlow. Follow this stackoverflow to resolve it.

"Hello World!" of deep learning

- Can computer recognize the following handwritten image numbers?

- The MNIST Dataset consists of 60,000 training images and 10,000 testing images. Each has a size of 28 x 28 pixel.

Thinking about images as data

- Images **are** data, but they are very unstructured
  - No instructions to say what is in them
  - No common grammar across images
  - Many, many possible subjects, objects, styles, etc.
- From a computer's perspective, images are just 3-dimensional matrices
  - Rows (pixels)
  - Columns (pixels)
  - Color channels (usually Red, Green, and Blue, ie, RGB)

- 798 rows
- 1200 columns
- 3 color channels
- $798 \times 1{,}200 \times 3 = 2{,}872{,}800$
  - The number of 'variables' per image like this!

Using images in practice

- There are a number of strategies to shrink images' dimensionality
  1. Downsample the image to a smaller resolution like 256x256x3
  2. Convert to grayscale
  3. Cut the image up and use sections of the image as variables instead of individual numbers in the matrix
     - Often done with convolutions in neural networks
  4. Drop variables that aren't needed, like LASSO

Image Reconition: MNIST Dataset

```
library(keras)

# MNIST dataset consists of 28 x 28 grayscale images of handwritten digits
mnist <- dataset_mnist()
str(mnist)
```

```
## List of 2
##  $ train:List of 2
##   ..$ x: int [1:60000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ y: int [1:60000(1d)] 5 0 4 1 9 2 1 3 1 4 ...
##  $ test :List of 2
##   ..$ x: int [1:10000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ y: int [1:10000(1d)] 7 2 1 0 4 1 4 9 5 9 ...
```

```
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
```

Image Reconition: MNIST Dataset

```
table(mnist$train$y, mnist$train$y)
```

```
##
##        0    1    2    3    4    5    6    7    8    9
##   0 5923    0    0    0    0    0    0    0    0    0
##   1    0 6742    0    0    0    0    0    0    0    0
##   2    0    0 5958    0    0    0    0    0    0    0
##   3    0    0    0 6131    0    0    0    0    0    0
##   4    0    0    0    0 5842    0    0    0    0    0
##   5    0    0    0    0    0 5421    0    0    0    0
##   6    0    0    0    0    0    0 5918    0    0    0
##   7    0    0    0    0    0    0    0 6265    0    0
##   8    0    0    0    0    0    0    0    0 5851    0
##   9    0    0    0    0    0    0    0    0    0 5949
```

Image Reconition: MNIST Dataset

```
table(mnist$test$y, mnist$test$y)
```

```
##
##        0    1    2    3    4    5    6    7    8    9
##   0  980    0    0    0    0    0    0    0    0    0
##   1    0 1135    0    0    0    0    0    0    0    0
##   2    0    0 1032    0    0    0    0    0    0    0
##   3    0    0    0 1010    0    0    0    0    0    0
##   4    0    0    0    0  982    0    0    0    0    0
##   5    0    0    0    0    0  892    0    0    0    0
##   6    0    0    0    0    0    0  958    0    0    0
##   7    0    0    0    0    0    0    0 1028    0    0
##   8    0    0    0    0    0    0    0    0  974    0
##   9    0    0    0    0    0    0    0    0    0 1009
```

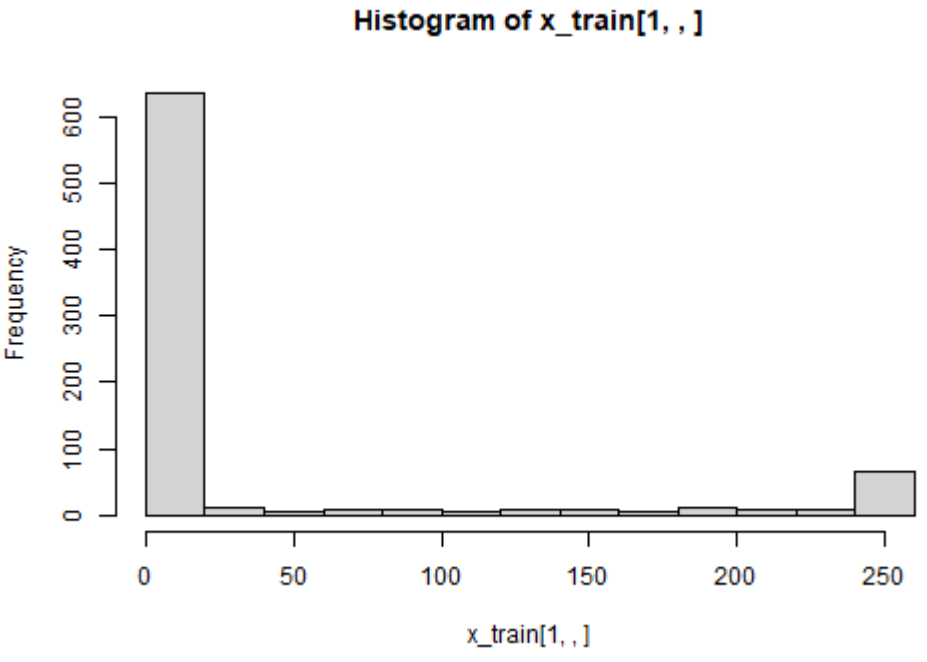
Image Reconition: MNIST Dataset

- as.raster(): create a raster object (representing a bitmap image)

```
par(mfrow = c(2,3)) # create 2x3 window to plot images
for (i in 1:6) plot(as.raster(x_train[i,,], max = 255))
```

Image Reconition: MNIST Dataset

```
hist(x_train[1,,])
```



Reshape and Rescale

- The x data is a 3-d array (images,width,height) of grayscale values. To prepare the data for training we convert the 3-d arrays into matrices by reshaping width and height into a single dimension (28x28 images are flattened into length 784 vectors). Then, we convert the grayscale values from integers ranging between 0 to 255 (0 black; 255 white) into floating point values ranging between 0 and 1:

```
# reshape
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
# rescale
x_train <- x_train / 255
x_test <- x_test / 255
str(x_train)
```

```
##  num [1:60000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
```

One-hot Encoding

- The `y` data is an integer vector with values ranging from 0 to 9. To prepare this data for training we one-hot encode (similar to dummy variables to represent categorical variables) the vectors into binary class matrices using the Keras `to_categorical()` function:

```
# One-hot encoding
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
head(y_train)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    0    0    1    0    0    0     0
## [2,]    1    0    0    0    0    0    0    0    0     0
## [3,]    0    0    0    0    1    0    0    0    0     0
## [4,]    0    1    0    0    0    0    0    0    0     0
## [5,]    0    0    0    0    0    0    0    0    0     1
## [6,]    0    0    1    0    0    0    0    0    0     0
```

Defining the Model

- There are two types of built-in models in Keras: `sequential models` and models created with the `functional API`. You can also create `custom models`.
- We use a sequential model and then adding layers:

```
# Define model
model <- keras_model_sequential() # specify the model
model %>%
  # first input layer with output units, must specify input shape c(28, 28)
  # the default activation is linear f(x) = x, we use ReLu function
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
  # fraction of input units to drop at each update during training time
  # help to prevent overfitting
  layer_dropout(rate = 0.4) %>%
  # one more hidden layer
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  # the output layer 10 numeric vectors
  # softmax function is to normalize numeric vectors into probabilities vectors
  layer_dense(units = 10, activation = 'softmax')
# how many layers in this model?
```

Model Summary

```
summary (model)
```

```
## Model: "sequential"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_2 (Dense)              (None, 256)               200960
## _________________________________________________________________
## dropout_1 (Dropout)          (None, 256)               0
## _________________________________________________________________
## dense_1 (Dense)              (None, 128)               32896
## _________________________________________________________________
## dropout (Dropout)            (None, 128)               0
## _________________________________________________________________
## dense (Dense)                (None, 10)                1290
## =================================================================
## Total params: 235,146
## Trainable params: 235,146
## Non-trainable params: 0
## _________________________________________________________________
```

```
# 200960 = 784 * 256 + 256
```

Compile Model

```
# Compile model
# optimizer: https://keras.io/optimizers/
# loss: https://keras.io/losses/
# metrics: https://keras.io/metrics/
model %>% compile(
  loss = 'categorical_crossentropy', # for multicategorical probalems
  optimizer = optimizer_rmsprop(), # optimizer for RecurrentNN
  metrics = c('accuracy') # prediction accuracy
)
```

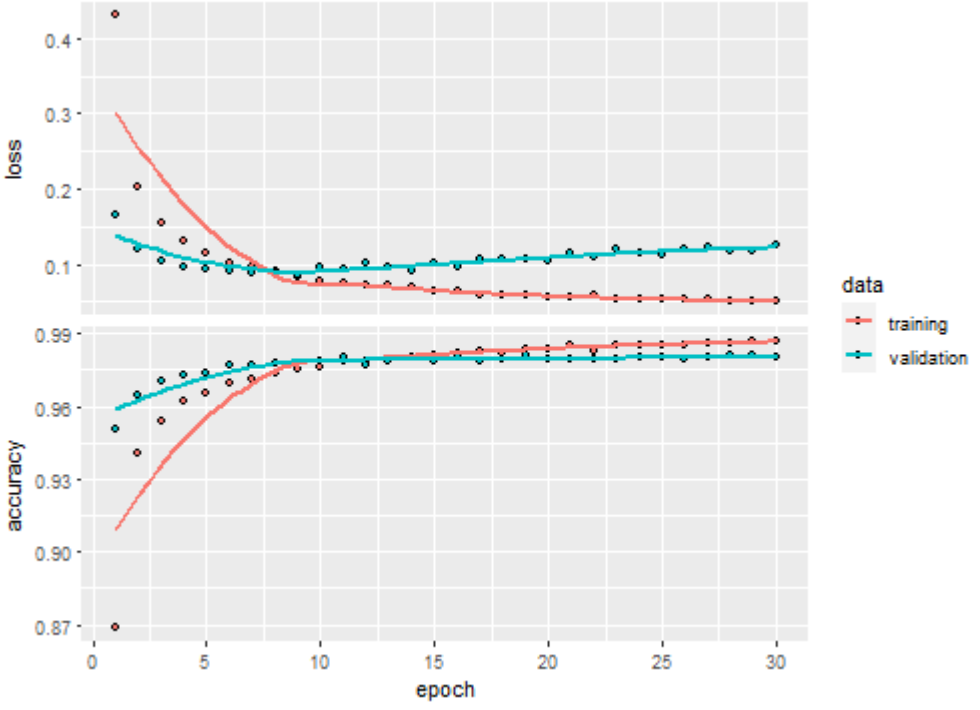
Training and Evaluation

```
# train the whole data in batches of 128 images, repeat for 30 times/epochs
history <- model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2 # % of training data to be used as validation data
)
```

- It will take some time or check the running log here

Plot Loss and Accuracy

```
plot(history)
```



Testing the Model: Loss and Accuracy

```
model %>% evaluate(x_test, y_test)
```

```
##      loss  accuracy
## 0.1027681 0.9818000
```

```
pred <- model %>% predict_classes(x_test)
pred[1:100]
```

```
##   [1] 7 2 1 0 4 1 4 9 6 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2 7
##  [38] 1 2 1 1 7 4 2 3 5 1 2 4 4 6 3 5 5 6 0 4 1 9 5 7 8 9 3 7 4 6 4 3 0 7 0 2 9
##  [75] 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 9 3 1 4 1 7 6 9
```

Testing the Model: Confusion matrix

```
table(Predicted = pred, Actual = mnist$test$y)
```

```
##           Actual
## Predicted    0    1    2    3    4    5    6    7    8    9
##         0  969    0    1    0    0    2    3    2    2    1
##         1    1 1126    1    0    0    0    2    1    2    2
##         2    0    4 1017    5    2    0    0    9    3    0
##         3    0    0    2  993    0    7    1    1    3    2
##         4    1    0    2    0  964    0    4    2    1    9
##         5    0    1    0    3    0  865    2    0    5    2
##         6    6    3    1    0    4   11  946    0    4    1
##         7    1    0    5    4    1    0    0 1006    4    5
##         8    1    1    3    3    2    3    0    1  945    0
##         9    1    0    0    2    9    4    0    6    5  987
```

Testing the Model: Probability

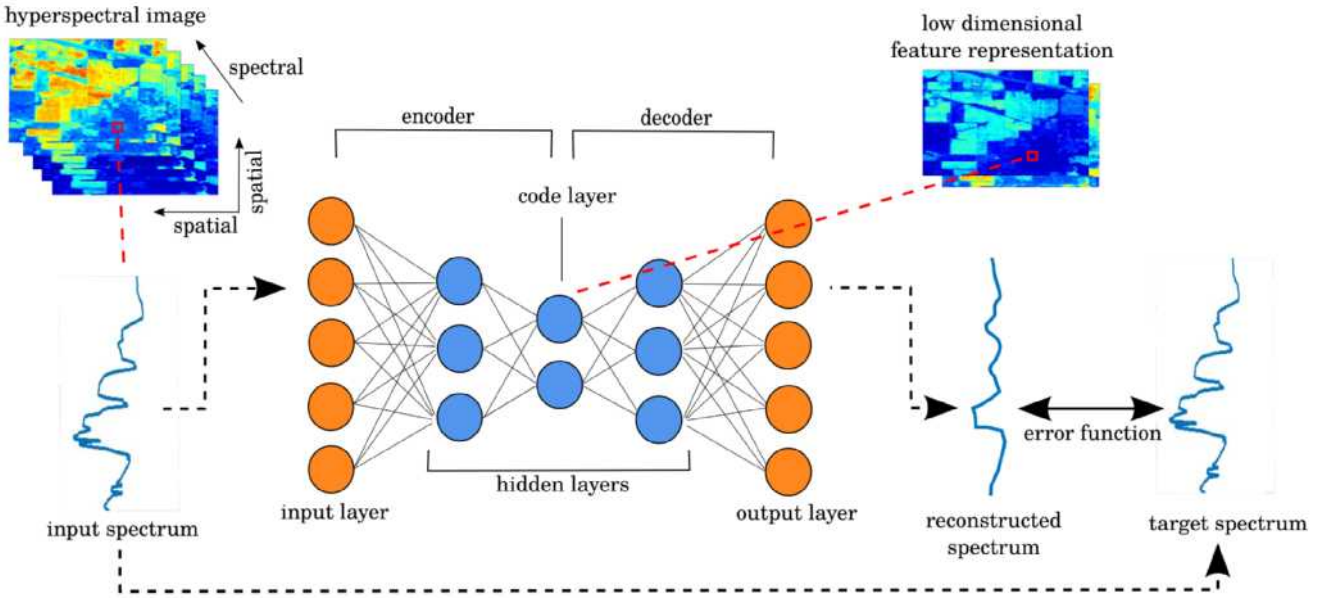
```
prob <- model %>% predict_proba(x_test)
cbind(prob, Predicted_class = pred, Actual = mnist$test$y)[1:5, ]
```

```
##
## [1,] 2.017509e-28 9.006533e-19 3.599984e-15 8.766629e-14 3.429586e-25
## [2,] 9.555513e-30 2.985998e-13 1.000000e+00 1.708584e-20 2.597416e-38
## [3,] 4.399435e-14 9.999998e-01 7.583802e-10 3.643570e-13 5.524010e-08
## [4,] 9.995081e-01 6.262326e-10 4.645616e-06 5.331166e-07 3.674779e-07
## [5,] 3.793510e-13 1.327507e-11 4.091135e-10 9.707870e-14 9.999911e-01
##
## [1,] 2.439239e-21 0.000000e+00 1.000000e+00 5.058997e-26 5.894975e-17
## [2,] 2.754897e-34 1.221704e-26 2.740914e-26 3.361469e-25 0.000000e+00
## [3,] 7.058971e-12 1.223351e-10 2.434750e-07 4.009426e-09 5.115570e-13
## [4,] 1.217527e-05 4.739283e-04 7.216580e-08 5.425399e-08 1.683024e-07
## [5,] 2.110876e-12 7.776806e-12 2.780264e-07 2.935277e-11 8.692212e-06
##      Predicted_class Actual
## [1,]               7      7
## [2,]               2      2
## [3,]               1      1
## [4,]               0      0
## [5,]               4      4
```

Fraud detection with autoencoder

What is autoencoder?

- Autoencoder (intro, advanced) is an unsupervised NN to reconstruct the original input through compressing and regenerating the data in the process.
  - Encoder: translate the original high-dimension input into the latent low-dimensional code (bottleneck). The input size is larger than the output size. Conceptually it is a nonlinear PCA dimentionality reduction process
  - Decoder: recover the data from the bottleneck.
  - E and D are NNs often with ReLU activation functions



How can autoencoder detect fraud/anomalies?

SMU SINGAPORE MANAGEMENT UNIVERSITY
School of Accountancy

> A well trained autoencoder model will produce a reconstructed output which mimics the input (ie, minimizing the recontruction loss). Observations with higher reconstruction loss could be identified as anomalies/fraud

Credit card fraud detection with autoencoder

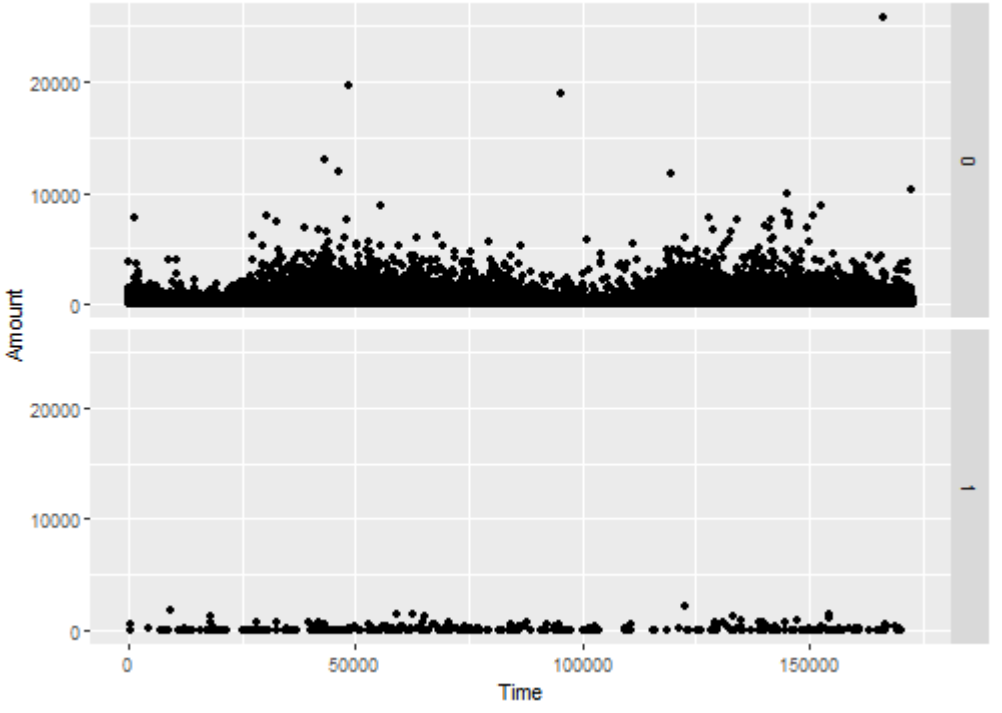
- The code is adapted from R Deep Learning Projects

```
# The dataset is from https://www.kaggle.com/mlg-ulb/creditcardfraud/data
df <- data.table::fread("../../Data/Session_11_creditcard.csv")
str(df)
```

```
## Classes 'data.table' and 'data.frame':    284807 obs. of  31 variables:
##  $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
##  $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
##  $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
##  $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
##  $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
##  $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
##  $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
##  $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
##  $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
##  $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
##  $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
##  $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
##  $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
##  $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
##  $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
##  $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
##  $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
##  $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
##  $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
```

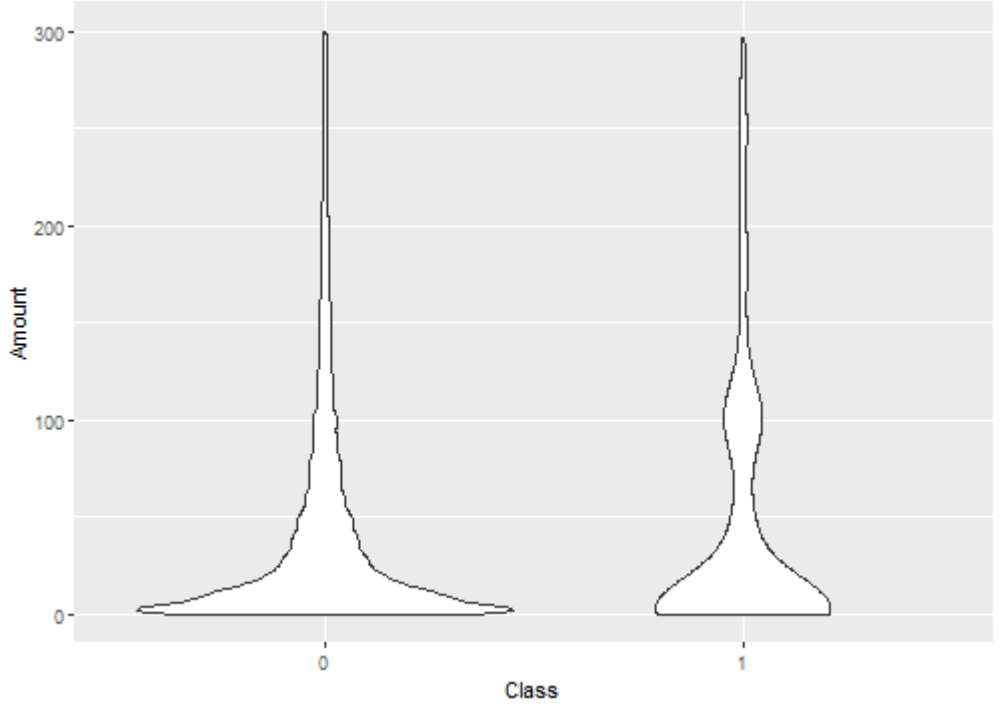
Any seasonality pattern

```
df %>% ggplot(aes(x = Time, y = Amount)) + geom_point() + facet_grid(Class ~ .)
```



Distribution by Class

```
# Look at smaller transactions
df$Class <- as.factor(df$Class)
df %>% filter(Amount<300) %>% ggplot(aes(x = Class, y = Amount)) + geom_violin()
```



Data preparation

```r
# Split the sample into train (90%) and test (10%)
# sample() to take a sample of the specified size
index <- sample(nrow(df), size = 0.1 * nrow(df))
train <- df[-index, ]
test <- df[index, ]

y_train <- train$Class
y_test <- test$Class

# Remove the Time and Class columns for x
x_train <- train %>% select(-c("Time", "Class"))
x_test <- test %>% select(-c("Time", "Class"))

# Coerce the dataframe to matrix to perform the training
# dataframe permits different data format
# matrix allows one data format only
# The hierarchy for coercion is: logical < integer < numeric < character
# as.matrix() https://rdrr.io/cran/data.table/man/as.matrix.html
x_train <- as.matrix(x_train)
x_test <- as.matrix(x_test)
```

Autoencoder preparation

```
# Autoencoder with the keras Functional API
# https://keras.rstudio.com/articles/functional_api.html
# Define parameters
input_dim <- 29 # 29 predictors/features
outer_layer_dim <- 14
inner_layer_dim <- 7
# input layer
input_layer <- layer_input(shape = c(input_dim))
# outputs compose input + dense layers
encoder <- input_layer %>%
  layer_dense(units = outer_layer_dim, activation = "relu") %>%
  layer_dense(units = inner_layer_dim, activation = "relu")
decoder <- encoder %>%
  layer_dense(units = inner_layer_dim) %>%
  layer_dense(units = outer_layer_dim) %>%
  layer_dense(units = input_dim)
```

Create model

```
# create model
autoencoder <- keras_model(inputs = input_layer, outputs = decoder)
autoencoder
```

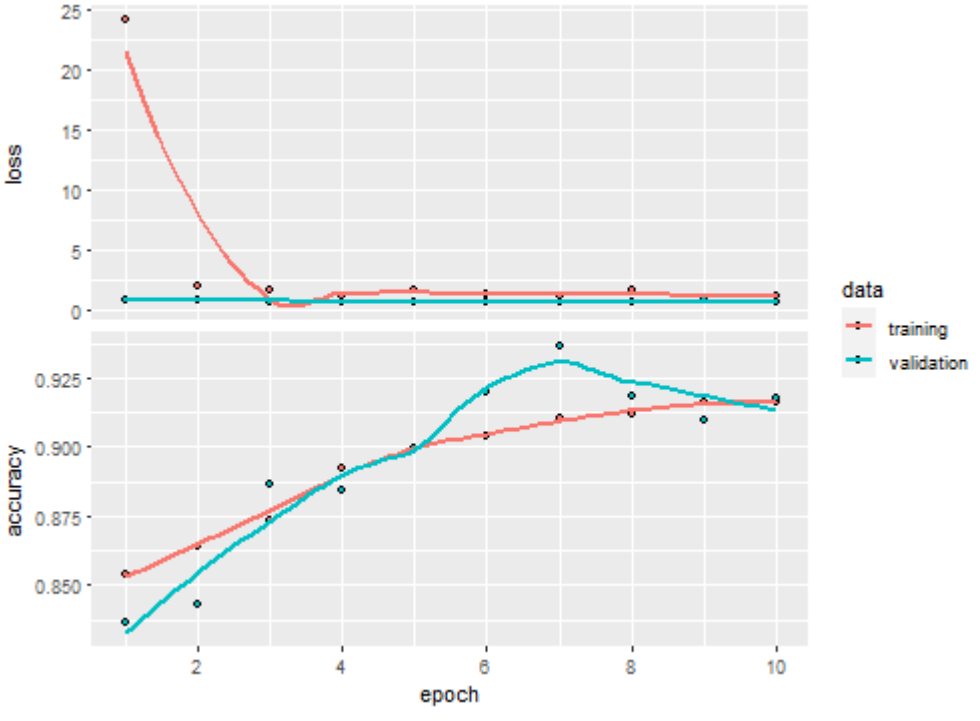
```
## Model
## Model: "functional_1"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## input_1 (InputLayer)         [(None, 29)]              0
## _________________________________________________________________
## dense_4 (Dense)              (None, 14)                420
## _________________________________________________________________
## dense_3 (Dense)              (None, 7)                 105
## _________________________________________________________________
## dense_7 (Dense)              (None, 7)                 56
## _________________________________________________________________
## dense_6 (Dense)              (None, 14)                112
## _________________________________________________________________
## dense_5 (Dense)              (None, 29)                435
## =================================================================
## Total params: 1,128
## Trainable params: 1,128
## Non-trainable params: 0
## _________________________________________________________________
```

Compile model

```
# compile model
# optimizer: https://keras.io/optimizers/
# loss: https://keras.io/losses/
# metrics: https://keras.io/metrics/
autoencoder %>% compile(
  optimizer = 'adam',
  loss = 'mean_squared_error',
  metrics = c('accuracy')
)
```

Fit model

```
# to save time, set 10 epochs only
history <- autoencoder %>% fit(x_train, x_train,  epochs = 10,
                               batch_size = 32, validation_split = 0.2)
plot(history)
```



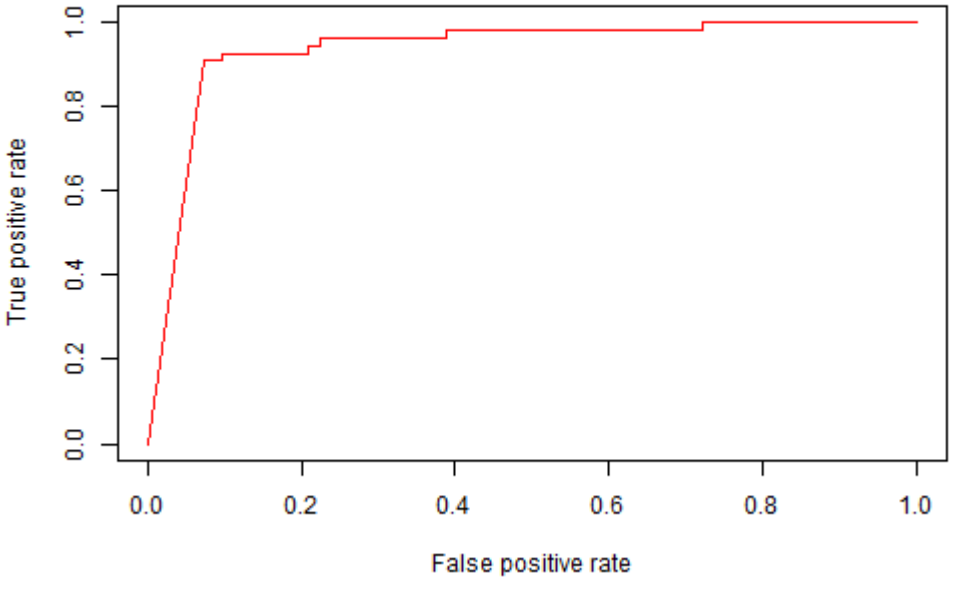
Model prediction

```
# Reconstruct on the test set or any other new dataset
# preds contains reconstructed/predicted values for all the 29 features
preds <- autoencoder %>% predict(x_test) %>% as.data.frame()

# As autoencoder is unsupervised algo, there is no prediction of Y
# predict Y based on reconstruction error
# for this case, fraud is predicted when sum of squared error SSE is >=30
# the threshold is not set in stone, you may try to vary it
# another way is to use quantile, say, the top 5% quantile to be anomaly
# the 95% quantile SSE is 32.3, use the following code to try
# threshold <- quantile(rowSums((preds - x_test)^2), probs = 0.95)
y_preds <- ifelse(rowSums((preds - x_test)^2) / 30 < 1,
                  rowSums((preds - x_test)^2) / 30, 1)
```

> You may also try the H2O package for deep learning (including autoencoder). The sample code is in the code file.

Model evaluation with AUC




```
## AUC of the autoencoder model
##                     0.9364324
```

Summary of Session 11

Recap

Today, we:

- Learned formally what neural networks (NNs) are
- Experienced the "Hello World!" version of deep learning using Keras and Tensoflow
  - Image recognition
- Apply neural networks to detect credit card fraud using Autoencoder

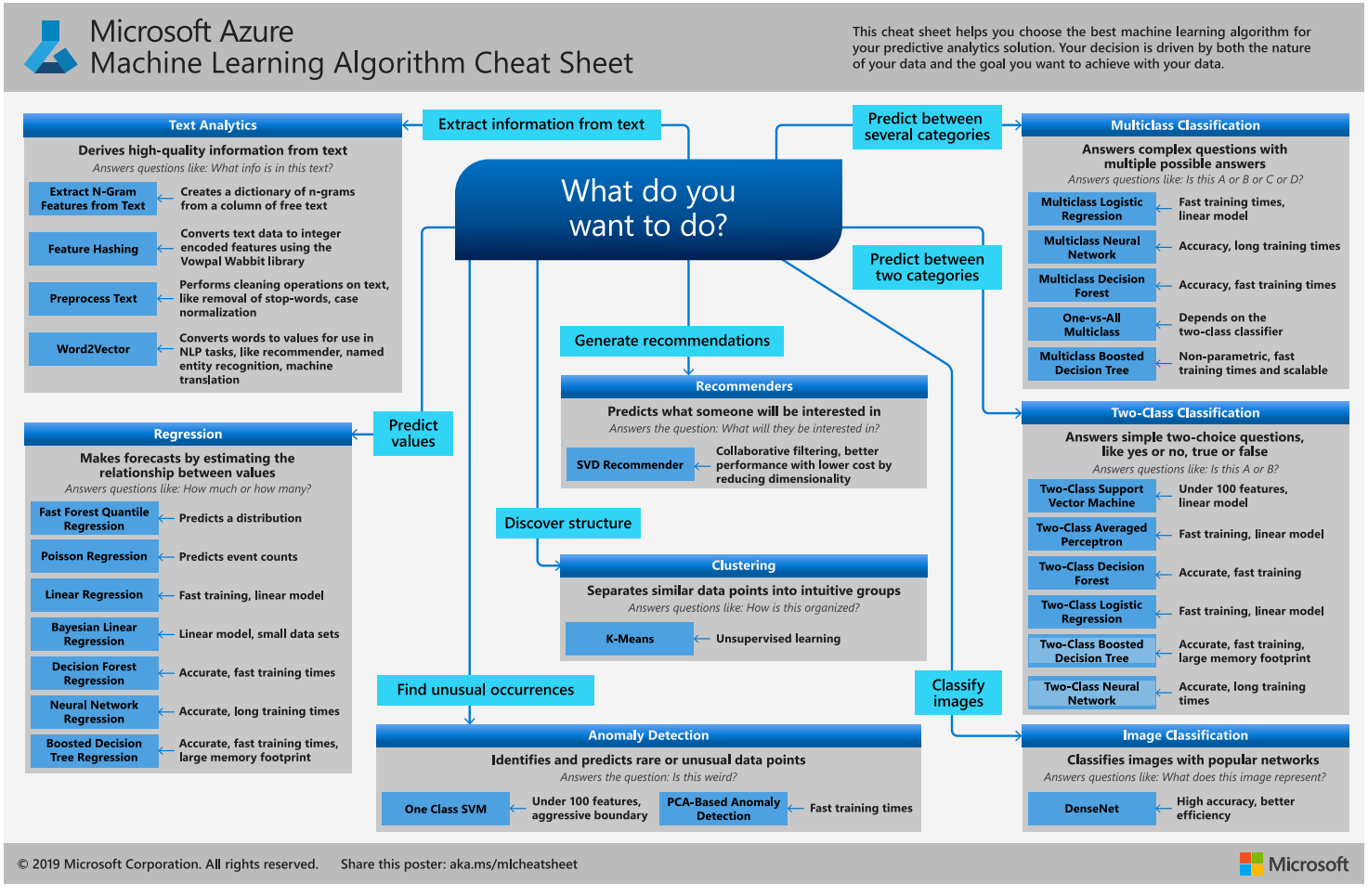
For next week

- Submit the group project and presentation slides by 1159pm, Sunday, 4 April 2021!
- Complete official class feedback form

> Groups 11 & 12 will be required to record your presentation in advance and submit to the instructor within 3 days after the last session. You are not required to do live online presentation in Session 12. You are strongly encouraged to attend the last session through ZOOM.

Summary of the Course

What did we learn?



R Coding Style Guide

SMU Singapore Management University School of Accountancy

- Style is subjective and arbitrary but it is important to follow a generally accepted style if you want to share code with others (or you want to read others' code)
- I suggest the The tidyverse style guide which is also adopted by Google with some modification
- Summary of **the tidyverse style guide**:
  - *File names*: end with .R
  - *Identifiers*: variable_name, function_name, try not to use . as it is reserved by Base R's S3 objects
  - *Line length*: 80 characters
  - *Indentation*: two spaces, no tabs (RStudio by default convert tabs to spaces)
  - *Spacing*: x = 0, not x=0, no space before a comma, but always place one after a comma
  - *Curly braces*: first on same line, last on own line
  - *Assignment*: use <-, not = nor ->
  - *Semicolons*: don't use, I used once for the interest of space
  - *return()*: Use explicit returns in functions: default function return is the last evaluated expression

SMU Singapore Management University
School of Accountancy

THANK YOU

R packages used in this slide

This slide was prepared on 2021-03-22 from Session_11s.Rmd with R version 4.0.3 (2020-10-10) Bunny-Wunnies Freak Out on Windows 10 x64 build 18362 😃.

The attached packages used in this slide are:

```
##       ROCR      keras    forcats    stringr      dplyr      purrr      readr
##   "1.0-11"  "2.3.0.0"    "0.5.1"    "1.4.0"    "1.0.4"    "0.3.4"    "1.4.0"
##      tidyr     tibble    ggplot2  tidyverse kableExtra      knitr
##    "1.1.2"    "3.0.6"    "3.3.3"    "1.3.0"    "1.1.0"     "1.31"
```