Week 11 Deep Learning

Theory and Practice

Ying Lin, Ph.D

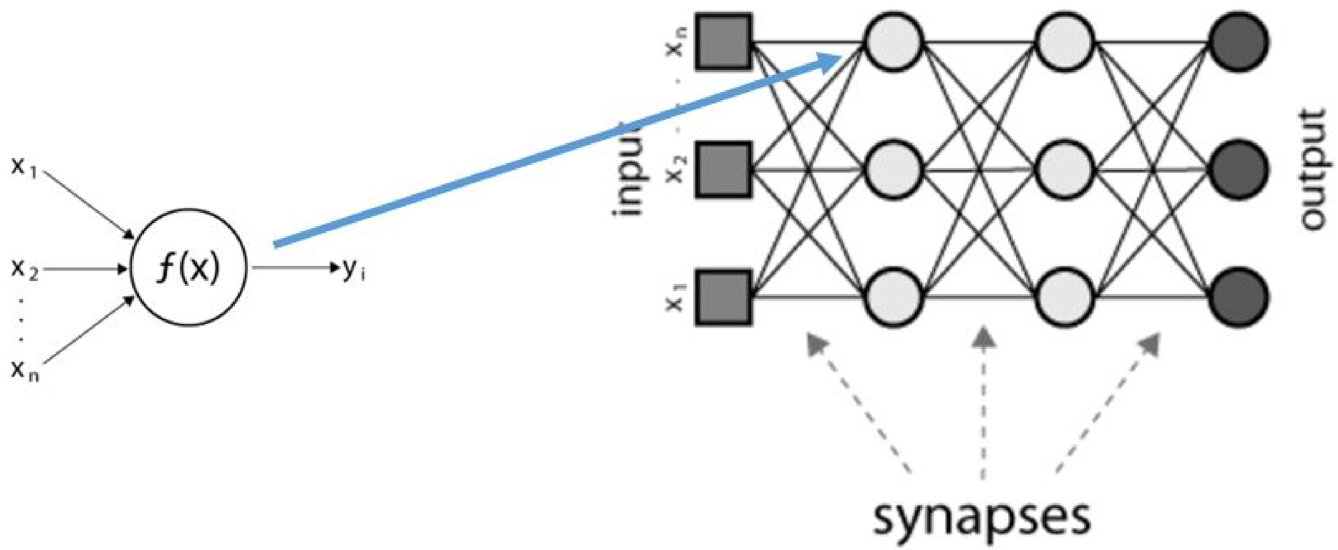
April 03, 2020

Outline

- Perceptron
- Deep learning model training and backpropagation algorithm
- Hyper-Parameters for deep learning model
- Different deep learning architectures
- R/Python demo

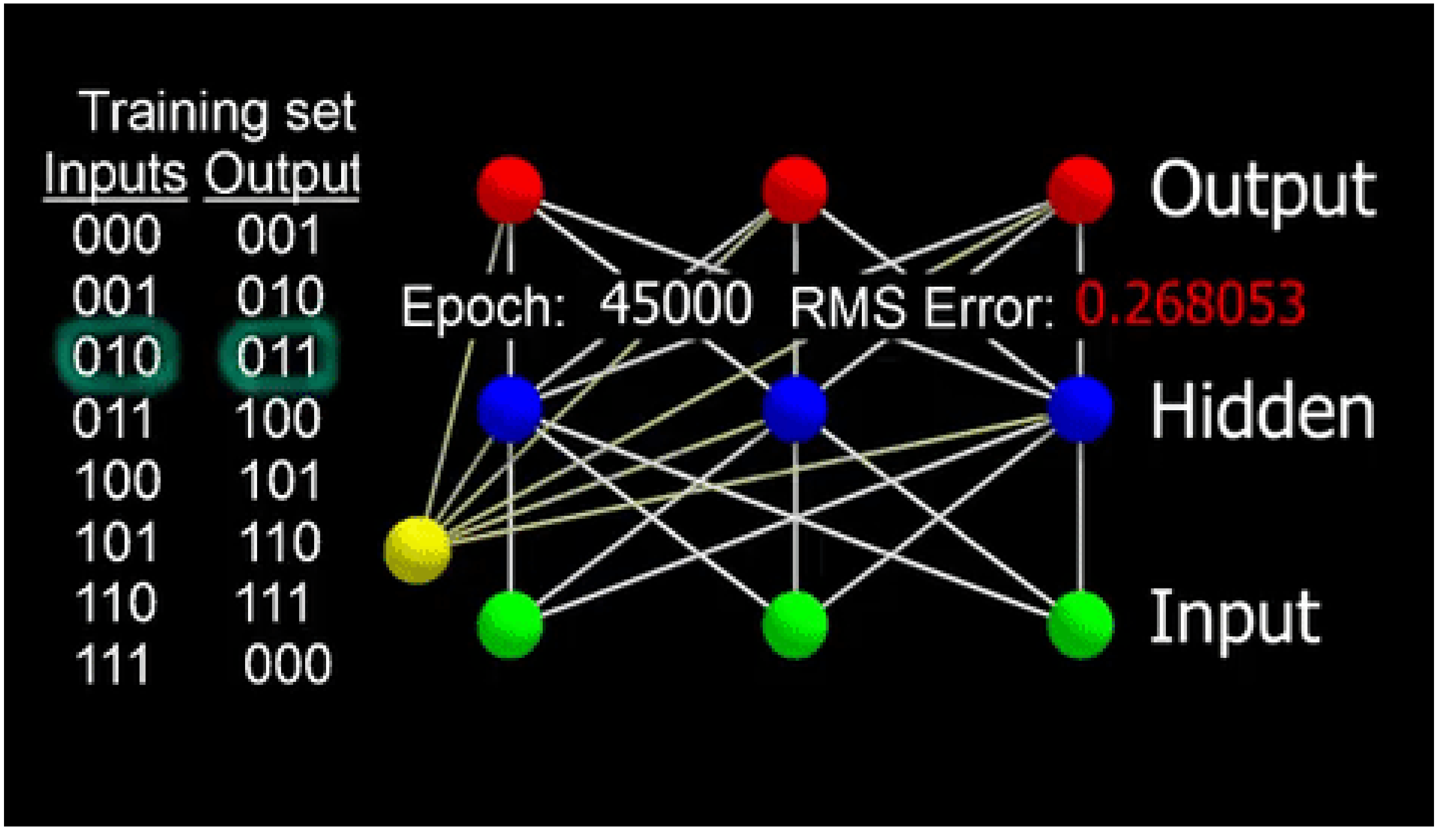
Motivations

- A node is a simple processing unit with linear or non-linear functions f
- Layers can have multiple nodes which are interconnected with nodes from other layers



3/41

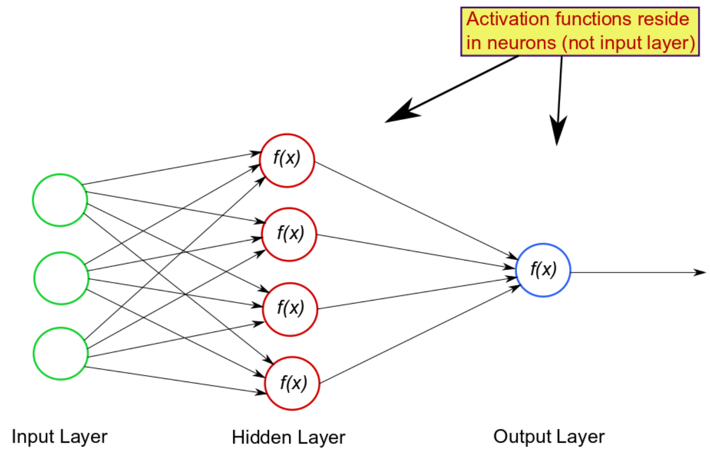
Visualization of ANN



4/41

Basic Neural Network Structure

- Nodes (neurons) that are organized in layers and the connections among nodes
  - Input layers
  - Hidden layers
  - Output layers
- A matrix of weights



5/41

Neural Network Architecture

- Multi-Layer Perceptron (MLP)
  - Fully-connected Neural Network
  - Feed-forward network
- Convolutional Neural Network (ConvNets or CNN)
- Recurrent Neural Network (RNN)
  - Long-Short Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)

Major Applications

- Computer vision: increase accuracy of ImageNet competition from ~84% to 95% in just three years
- Natural language processing
- AlphaGO
- Autonomous car

Data Preprocessing

- Convert categorical into numerical
- Normalization: to avoid death of those weights associated with input nodes (input variables) with smaller scale and the whole network won't update from epoch and epoch
- Missing value imputation
- Variable multicollinearity is not an issue
- Embedding: low-dimensional representations

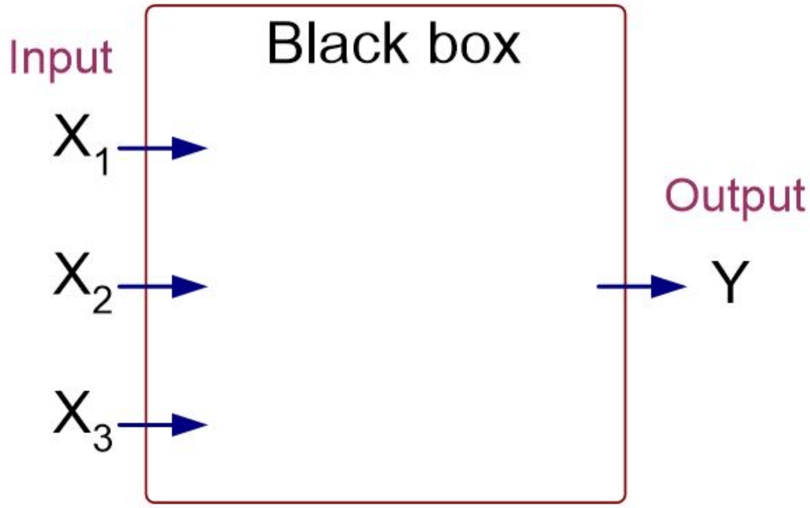
Perceptron

- Perceptron is the most basic neuron unit
- Input layer
- Output layer
  - Linear combination of input nodes $\boldsymbol{X}$ and bias node x_0
  - Apply activation function $f(x)$
  - $\boldsymbol{Y} = f(\beta \boldsymbol{X})$
- No hidden layer
- Connection between perceptron and logistic regression
  - With sigmoid function as activiation function, perceptron is the same as logistic regression $\boldsymbol{Y} = \frac{1}{1+e^{-\beta X}}$ which is the same as modeling the probability for a binomial RV.

9/41

Example of Perceptron

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



10/41

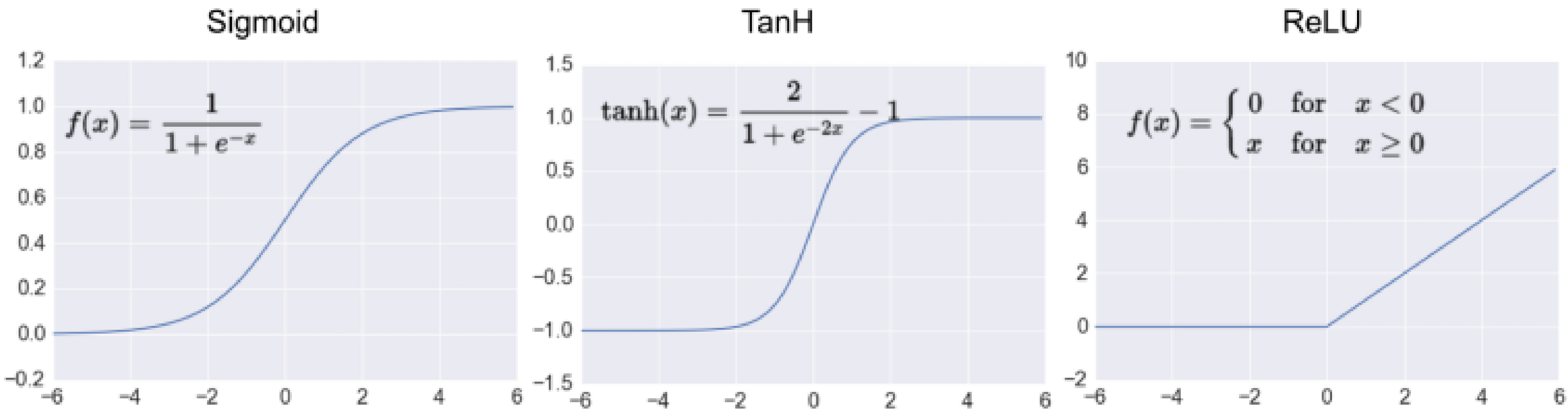
Different Types of Activation Function

- Enable neural network model to capture the non-linearity relationship between data
- Sigmoid function: “squashes” output values to the range of 0 and 1
  - $s'(x) = s(x)(1 - s(x))$
  - Output does not center around 0
  - Varnishing gradient
- $\tanh$ function: hyperbolic tangent function
  - Similar to sigmoid function but “squashes” ouptut to (-1, 1)
  - $tanh(x) = 2S(2x) - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
  - Scaled or shifted sigmoid and often preferable to Sigmoid function

Different Type of Activation Function (2)

- ReLU (Rectified Linear Unit) function $f(x) = \max(0, x)$
  - Shown to accelerate convergence of gradient descent compared to other functions
  - Default activation function for hidden layers
  - Derivative of ReLU at $x = 0$ is undefined; we set it to 0
- Requirement of activation function
  - Continuously differentiable: needed for gradient-based optimization method
  - Nonlinear: allows the neural network to be a universal approximation

Different Type of Activation Function (3)



13/41

Softmax Function

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

- z is a vector of the inputs to the output layer
- j index the output layers
- Softmax function squashes the outputs to (0, 1) similar to sigmoid function; signmod function is a special case of softmax function when # of classes = 2
- Sum of outputs add up to 1; therefore equivalent to a categorical probability distribution
- Often used for multinomial classification

Loss/Cost Functions (1)

- Required characteristics of cost function
  - Must be continuously differentiable with respect to model parameters
  - Concave vs. convex function: global optimal
- Sum of squared error

$$Loss(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

15/41

Loss/Cost Functions (2)

- Binary cross entropy/log loss function

$$Loss(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}(y_i log\hat{y}_i + (1 - y_i)log(1 - \hat{y}_i))$$

$$\frac{\partial Loss}{\partial a^L} = \frac{a^L - y}{a^L(1 - a^L)}$$

- Categorical cross entropy loss

$$L(y, \hat{y}) = -\sum_{i=1}^{N}\sum_{j=1}^{M}(y_{ij} * log(\hat{y}_{ij}))$$

- Cross entropy loss for categorical attribute
  - Compare how well the probability distribution output by softmax matches the one-hot-encoded ground truth label of the data

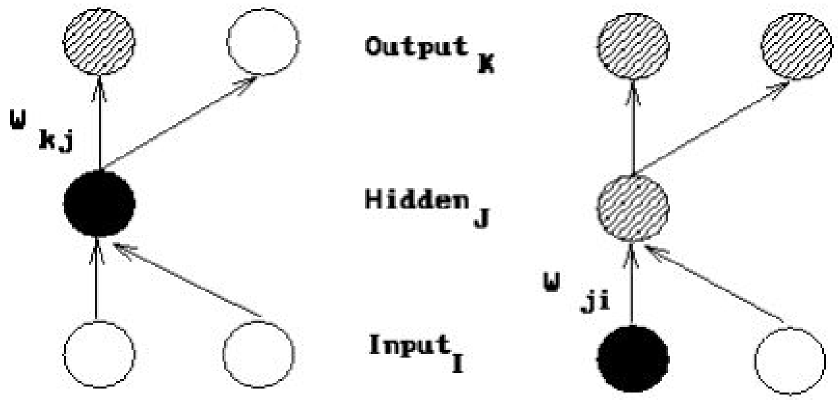
16/41

Backpropagation

- Random initialization of weights for connections
- Forward propagation of weight
- Compare predicted outputs against real outputs
- Back propagate the errors through the neural network
- Use weight update rule to update weights $w_{i+1} = w_i + \eta \frac{\partial \delta}{\partial w_i}$
- Example of using backpropagation algorithm and delta update rules to train a neural network model
  - use a small network with 1 hidder layer and two nodes per layer to demonstrate how to update network weights

17/41

Math Notation



- k for output layer; j for hidden layer; i for input layer
- w_{kj} denotes a weight from the hidden to the output layer
- w_{ji} denotes a weight from the input to the hidden layer
- a denotes an activation value
- t denotes a target value
- net denotes the net input

18/41

Review of Derivative Rules from Calculus

- Chain rule: $\frac{d(e^u)}{dx} = e^u \frac{du}{dx}$
- Sum rule: $\frac{d(g+h)}{dx} = \frac{dg}{dx} + \frac{dh}{dx}$
- Power rule: $\frac{d(g^n)}{dx} = ng^{n-1} \frac{dg}{dx}$
- Derivative of sigmoid function ($s(x) = \frac{1}{1+e^{-x}}$)
  - $s'(x) = s(x)(1 - s(x))$

19/41

Weight Update for a Hidden to Output Weight

$$\Delta w_{kj} \propto -\frac{\partial E}{\partial w_{kj}} = -\epsilon \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial net_k} \frac{\partial net_k}{\partial w_{jk}}$$

- derivative of the squared error with respect to the activation
  - $\frac{\partial E}{\partial a_k} = \frac{\partial(\frac{1}{2}(t_k - a_k)^2)}{\partial a_k} = -(t_k - a_k)$
- derivative of the activation with respect to the net input
  - $\frac{\partial a_k}{\partial net_k} = a_k(1 - a_k)$
- derivative of the net input with respect to a weight
  - $\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial(w_{kj} a_j)}{\partial w_{kj}} = a_j$
- to sum up
  - $\Delta w_{kj} = \epsilon(t_k - a_k)a_k(1 - a_k)a_j = \epsilon\delta_k a_j$

Weight Change Rule for an Input to Hidden Weight

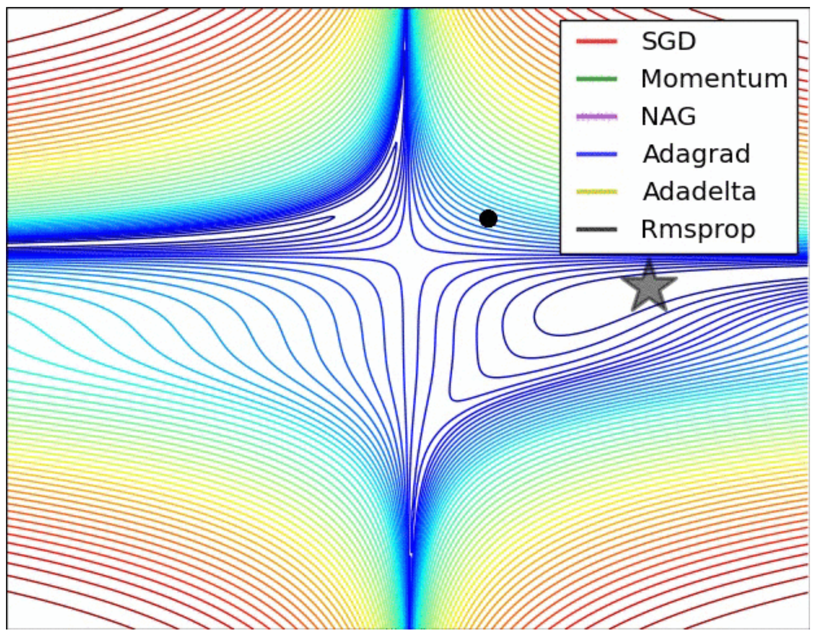
$$
\begin{aligned}
\Delta w_{ji} &\propto -[\sum_k \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial net_k}\frac{\partial net_k}{\partial a_j}]\frac{\partial a_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} \\
&= \epsilon[\sum_k (t_k - a_k)a_k(1-a_k)w_{kj}]a_j(1-a_j)a_i \\
&= \epsilon[\sum_k \delta_k w_{kj}]a_j(1-a_j)a_i \\
&= \epsilon\delta_j a_i
\end{aligned}
$$

Model Hyper-Parameters to Tune in Deep Learning

- Number of hidden layers
- Number of nodes in each layer
- Loss function
- Activation function
- Optimization method (e.g. gradient descent)
- Epoch
- Mini-batch or individual update weights
- Regularization: L1/L2 form, dropout rate, etc.

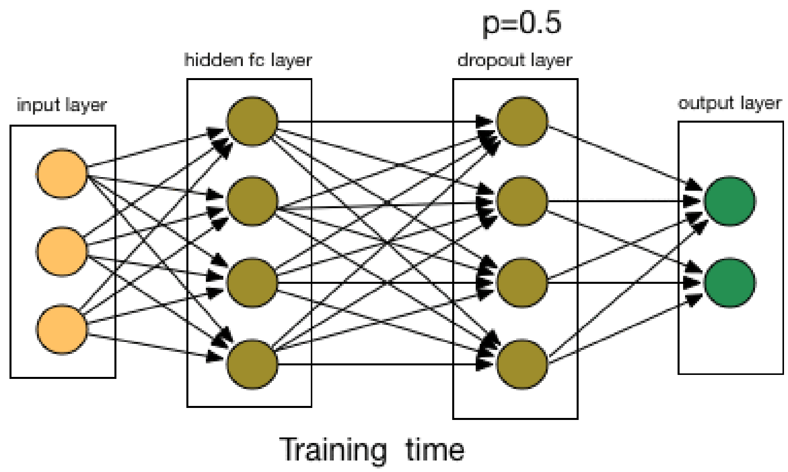
Optimization Algorithm

- Gradient descent and its variants
  - Stochastic gradient descent (SGD)
  - Mini batch gradient descent
- Adam (Adaptive Moment Estimation): works well in practice



Regularization in Deep Learning

- Dropout



- Early stop
- L1/L2 regularization: add regularization term to the loss function

24/41

Characteristics of Deep Learning

- Universal approximator with a very expressive hypothesis space
- Tolerate redundant features
- Sensitive towards noise
- Gradient descent method converges to local minimum
- Training is expensive while testing is fast

Demo Data: Customer Churn

```
churn_data_raw <- read.csv("data/Telco-Customer-Churn.csv")
str(churn_data_raw)
```

```
## 'data.frame':    7043 obs. of  21 variables:
##  $ customerID      : Factor w/ 7043 levels "0002-ORFBO","0003-MKNFE",..: 5376 3963 2565 5536
##  $ gender          : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 1 2 1 1 2 ...
##  $ SeniorCitizen   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Partner         : Factor w/ 2 levels "No","Yes": 2 1 1 1 1 1 1 2 1 ...
##  $ Dependents      : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 2 1 1 2 ...
##  $ tenure          : int  1 34 2 45 2 8 22 10 28 62 ...
##  $ PhoneService    : Factor w/ 2 levels "No","Yes": 1 2 2 1 2 2 2 1 2 2 ...
##  $ MultipleLines   : Factor w/ 3 levels "No","No phone service",..: 2 1 1 2 1 3 3 2 3 1 ...
##  $ InternetService : Factor w/ 3 levels "DSL","Fiber optic",..: 1 1 1 1 2 2 2 1 2 1 ...
##  $ OnlineSecurity  : Factor w/ 3 levels "No","No internet service",..: 1 3 3 3 1 1 1 3 1 3 .
##  $ OnlineBackup    : Factor w/ 3 levels "No","No internet service",..: 3 1 3 1 1 1 3 1 1 3 .
##  $ DeviceProtection: Factor w/ 3 levels "No","No internet service",..: 1 3 1 3 1 3 1 1 3 1 .
##  $ TechSupport     : Factor w/ 3 levels "No","No internet service",..: 1 1 1 3 1 1 1 1 3 1 .
##  $ StreamingTV     : Factor w/ 3 levels "No","No internet service",..: 1 1 1 1 1 3 3 1 3 1 .
##  $ StreamingMovies : Factor w/ 3 levels "No","No internet service",..: 1 1 1 1 1 3 1 1 3 1 .
##  $ Contract        : Factor w/ 3 levels "Month-to-month",..: 1 2 1 2 1 1 1 1 1 2 ...
##  $ PaperlessBilling: Factor w/ 2 levels "No","Yes": 2 1 2 1 2 2 2 1 2 1 ...
##  $ PaymentMethod   : Factor w/ 4 levels "Bank transfer (automatic)",..: 3 4 4 1 3 3 2 4 3 1
##  $ MonthlyCharges  : num  29.9 57 53.9 42.3 70.7 ...
```

Data Splitting

```
library(tidyverse)
library(rsample)
churn_data_tbl <- churn_data_raw %>%
  select(-customerID) %>%
  drop_na() %>%
  select(Churn, everything())
set.seed(100)
train_test_split <- initial_split(churn_data_tbl, prop = 0.8)
train_tbl <- training(train_test_split)
test_tbl <- testing(train_test_split)
```

Data Preprocessing (1)

```
library(recipes)
rec_obj <- recipe(Churn ~ ., data = train_tbl) %>%
  step_discretize(tenure, options = list(cuts = 6)) %>%
  step_log(TotalCharges) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_center(all_predictors(), -all_outcomes()) %>%
  step_scale(all_predictors(), -all_outcomes()) %>%
  prep(data = train_tbl)
```

Data Preprocessing (2)

```
rec_obj
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         19
##
## Training data contained 5626 data points and no missing data.
##
## Operations:
##
## Dummy variables from tenure [trained]
## Log transformation on TotalCharges [trained]
## Dummy variables from gender, Partner, Dependents, tenure, ... [trained]
## Centering for SeniorCitizen, ... [trained]
## Scaling for SeniorCitizen, ... [trained]
```

29/41

Data Preprocessing (3)

```
x_train_tbl <- bake(rec_obj, newdata = train_tbl) %>% select(-Churn)
x_test_tbl <- bake(rec_obj, newdata = test_tbl) %>% select(-Churn)
y_train_vec <- ifelse(pull(train_tbl, Churn) == "Yes", 1, 0)
y_test_vec <-  ifelse(pull(test_tbl, Churn) == "Yes", 1, 0)
str(x_train_tbl)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    5626 obs. of  35 variables:
##  $ SeniorCitizen                          : num  -0.435 -0.435 -0.435 -0.435 -0.435 ...
##  $ MonthlyCharges                         : num  -1.158 -0.26 -0.745 0.195 1.154 ...
##  $ TotalCharges                           : num  -2.276 0.389 0.372 -1.231 -0.147 ...
##  $ gender_Male                            : num  -1.002 0.998 0.998 -1.002 -1.002 ...
##  $ Partner_Yes                            : num  1.026 -0.974 -0.974 -0.974 -0.974 ...
##  $ Dependents_Yes                         : num  -0.651 -0.651 -0.651 -0.651 -0.651 ...
##  $ tenure_bin1                            : num  2.168 -0.461 -0.461 2.168 -0.461 ...
##  $ tenure_bin2                            : num  -0.439 -0.439 -0.439 -0.439 2.278 ...
##  $ tenure_bin3                            : num  -0.448 -0.448 -0.448 -0.448 -0.448 ...
##  $ tenure_bin4                            : num  -0.451 2.217 2.217 -0.451 -0.451 ...
##  $ tenure_bin5                            : num  -0.45 -0.45 -0.45 -0.45 -0.45 ...
##  $ tenure_bin6                            : num  -0.434 -0.434 -0.434 -0.434 -0.434 ...
##  $ PhoneService_Yes                       : num  -3.041 0.329 -3.041 0.329 0.329 ...
##  $ MultipleLines_No.phone.service         : num  3.041 -0.329 3.041 -0.329 -0.329 ...
##  $ MultipleLines_Yes                      : num  -0.857 -0.857 -0.857 -0.857 1.166 ...
##  $ InternetService_Fiber.optic            : num  -0.888 -0.888 -0.888 1.125 1.125 ...
```

Deep Learning Model Specification

```
library(keras)
model_keras <- keras_model_sequential()
model_keras %>%
  layer_dense(units = 16,
    kernel_initializer = "uniform",
    activation = "relu",
    input_shape = ncol(x_train_tbl)) %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 16,
    kernel_initializer = "uniform",
    activation = "relu") %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 1,
    kernel_initializer = "uniform",
    activation = "sigmoid") %>%
  compile(optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = c("accuracy")
  )
```

Model Details

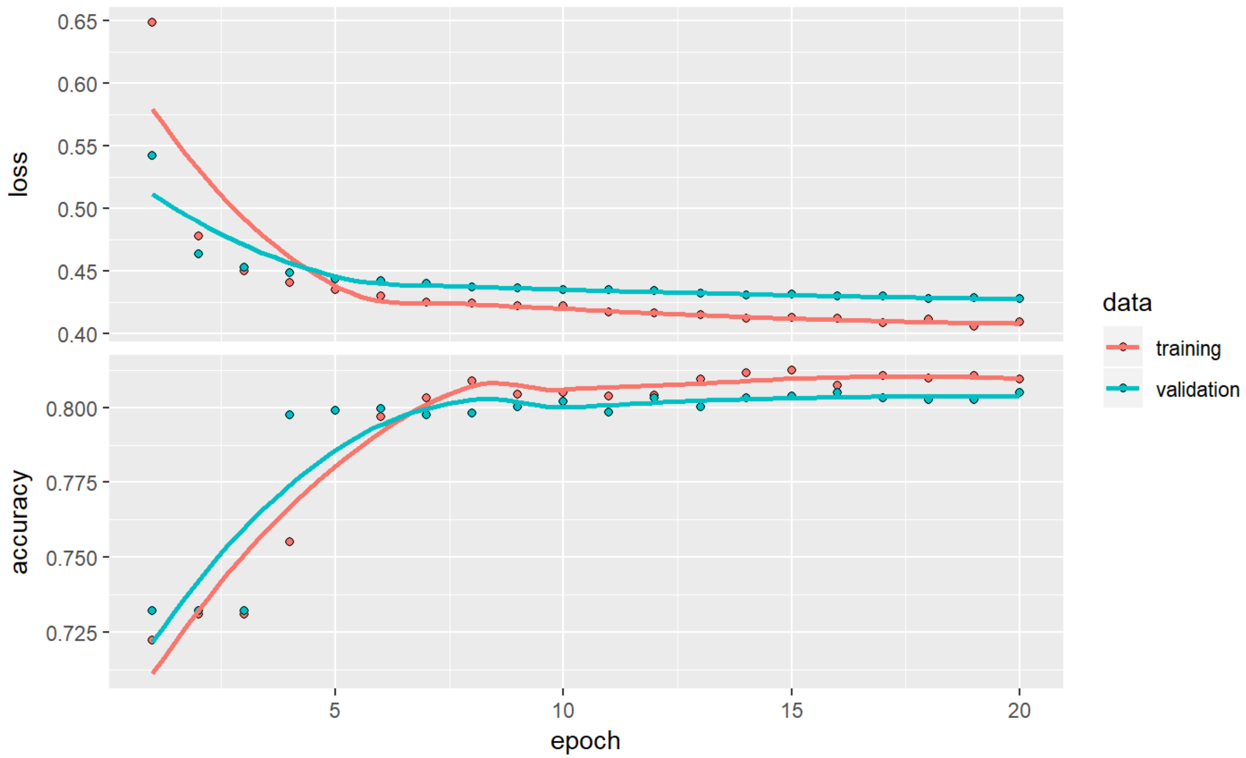
```
model_keras
```

```
## Model
## Model: "sequential_1"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## dense_1 (Dense)                     (None, 16)                      576
## ________________________________________________________________________________
## dropout_1 (Dropout)                 (None, 16)                      0
## ________________________________________________________________________________
## dense_2 (Dense)                     (None, 16)                      272
## ________________________________________________________________________________
## dropout_2 (Dropout)                 (None, 16)                      0
## ________________________________________________________________________________
## dense_3 (Dense)                     (None, 1)                       17
## ================================================================================
## Total params: 865
## Trainable params: 865
## Non-trainable params: 0
## ________________________________________________________________________________
```

Model Training

Model Training Process

```
plot(history)
```



33/41

Model Prediction

```
yhat_keras_class_vec <- predict_classes(object = model_keras, x = as.matrix(x_test_tbl)) %>%
  as.vector()
yhat_keras_prob_vec <- predict_proba(object = model_keras, x = as.matrix(x_test_tbl)) %>%
  as.vector()
estimates_keras_tbl <- tibble(
  truth = as.factor(y_test_vec) %>% fct_recode(yes = "1", no = "0"),
  estimate = as.factor(yhat_keras_class_vec) %>% fct_recode(yes = "1", no = "0"),
  class_prob = yhat_keras_prob_vec
)
estimates_keras_tbl
```

```
## # A tibble: 1,406 x 3
##    truth estimate class_prob
##    <fct> <fct>         <dbl>
##  1 yes   no          0.382
##  2 yes   yes         0.625
##  3 no    no          0.00784
##  4 no    no          0.00938
##  5 no    no          0.0259
##  6 no    no          0.113
##  7 no    yes         0.698
##  8 no    no          0.495
##  9 no    no          0.0151
```

Model Performance Evaluation (1)

```
library(yardstick)
options(yardstick.event_first = F)
estimates_keras_tbl %>% conf_mat(truth, estimate)
```

```
##           Truth
## Prediction  no yes
##        no  945 153
##        yes 104 204
```

```
estimates_keras_tbl %>% metrics(truth, estimate)
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.817
## 2 kap      binary         0.495
```

Model Performance Evaluation (2)

```
estimates_keras_tbl %>% roc_auc(truth, class_prob)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.848
```

```
estimates_keras_tbl %>% precision(truth, estimate)
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.662
```

```
estimates_keras_tbl %>% recall(truth, estimate)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  binary         0.571
```

Load Python Packages

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, cross_vali
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot as plt
np.random.seed(66)
```

Load and Prepare the Datasets

```
churn = pd.read_csv('churn.csv')
churn['international plan'] = churn['international plan'].map(dict(yes=1, no=0))
churn['voice mail plan'] = churn['voice mail plan'].map(dict(yes=1, no=0))

num_vars = churn.select_dtypes(['int64', 'float64']).columns
X = churn[num_vars]
y = churn.churn

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=16)
X_test.shape
```

```
## (1000, 18)
```

```
y_train.shape
```

```
## (2333,)
```

Attribute Normalization and Standardization

```
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
X_train.shape
```

```
## (2333, 18)
```

Deep Learning Model Architecture

```
from keras import models
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(6, activation='relu', input_shape=(18, )))
model.add(Dense(6, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.output_shape
```

```
## (None, 1)
```

```
model.summary()
```

```
## Model: "sequential_2"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_4 (Dense)              (None, 6)                 114
## _________________________________________________________________
## dense_5 (Dense)              (None, 6)                 42
## _________________________________________________________________
```

Model Specification and Training

```
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=20, batch_size=1, verbose=1)
```

```
## Epoch 1/20
##
##    1/2333 [..............................] - ETA: 2:42 - loss: 0.7894 - accuracy: 0.0000e+0
##   79/2333 [>.............................] - ETA: 3s - loss: 0.7432 - accuracy: 0.5570
##  155/2333 [>.............................] - ETA: 2s - loss: 0.6950 - accuracy: 0.6516
##  223/2333 [=>............................] - ETA: 2s - loss: 0.6362 - accuracy: 0.7175
##  285/2333 [==>...........................] - ETA: 1s - loss: 0.6181 - accuracy: 0.7404
##  373/2333 [===>..........................] - ETA: 1s - loss: 0.5820 - accuracy: 0.7721
##  459/2333 [====>.........................] - ETA: 1s - loss: 0.5514 - accuracy: 0.7952
##  544/2333 [=====>........................] - ETA: 1s - loss: 0.5318 - accuracy: 0.8051
##  633/2333 [=======>......................] - ETA: 1s - loss: 0.5208 - accuracy: 0.8120
##  719/2333 [========>.....................] - ETA: 1s - loss: 0.5148 - accuracy: 0.8122
##  797/2333 [=========>....................] - ETA: 1s - loss: 0.5128 - accuracy: 0.8118
##  880/2333 [==========>...................] - ETA: 1s - loss: 0.5000 - accuracy: 0.8182
##  962/2333 [===========>..................] - ETA: 0s - loss: 0.4902 - accuracy: 0.8222
## 1050/2333 [============>.................] - ETA: 0s - loss: 0.4780 - accuracy: 0.8286
## 1140/2333 [=============>................] - ETA: 0s - loss: 0.4746 - accuracy: 0.8307
## 1230/2333 [==============>...............] - ETA: 0s - loss: 0.4616 - accuracy: 0.8358
## 1317/2333 [===============>..............] - ETA: 0s - loss: 0.4557 - accuracy: 0.8383
## 1408/2333 [=================>............] - ETA: 0s - loss: 0.4557 - accuracy: 0.8366
```