Wooyong Jung
Net ID: wj710

# Predicting Economic Performance from Countries' Staple Products

## 1. Introduction

Investigating countries' product specialization can have important implications for understanding their overall economy. Because, in order to produce goods, a country has to use its labor, accumulated skills, and natural resources along with its own culture and institutions produce goods, the staple products can be regarded as the final result of integrating all physical and human capital that the country has (Hausmann et al. 2007). In this light, studying which products a country is specialized in can be a good predictor of many other socioeconomic symptoms we observed.

This study investigated how much a country's staple products can predict its economic growth and total factor productivity (TFP), using a Neural Network method. Since annual economic growth reflects internal and external stimulus well and can be quantified relatively easily, it has been widely used to represent countries' socioeconomic performance. TFP has been also used in many studies to measure technology growth and efficiency in a country. Thus, as alternately using these two indicators as a predicted target, this study tested how accurately countries' staple products can predict their economic performance.

## 2. Data

The primary purpose of this study is to identify how well the group of products in which a country is specialized predict its economic growth and TFP. To this end, this study combined trade, economic growth, and TFP datasets. The details of each dataset are as follows.

*Input*

In order to capture the items each country has had a specialization to produce in each year, this study used the revealed comparative advantage (RCA) which was first devised by Balassa (1965). If a country's RCA in a specific product in a specific year is greater than 1, it means that the share of the country's exports in that product is larger than the share of the product in the world export in that year. It can be calculated by

$$RCA_{pct} = \frac{\frac{EX_{pct}}{\sum_p EX_{pct}}}{\frac{\sum_p EX_{pct}}{\sum_c \sum_p EX_{cit}}}$$

where p is a product, c is a country, t is a year, and EX is the amount of export. The Atlas of Economic Complexity operated by Harvard's Growth Lab has provided the RCA values based on two trade classification systems: Standard International Trade Classification (SITC) and Harmonized System (HS). This study used SITC 4-Code (rev.2) considering it covers a longer period than the HS 4-Code (1962 to 2017). The number of total products classified in SITC 4-Code is 782. Since the RCA varies to a large extent across country and product, this study converted it to a binary variable; 1 for a product that country has a comparative advantage (RCA > 1), and 0 otherwise.

*Output*

For economic growth, this study used the GDP growth based on constant 2010 U.S dollars from the World Bank national accounts data. In order for the prediction to be a classification matter, this study classified the growth rates into 4 levels based on its quantile. For measuring TFP, this study borrowed the TFP level data which was measured at current PPPs from the Penn World Table Version 9.1. As with the GDP growth rate, TFP was also classified into 4 levels. The thresholds of each quantile of growth rate and TFP are as is Table 1.

Table 1. Thresholds of Growth and TFP Level

| Quantile | 1 – 25th | 26 – 50th | 51 – 75th | 76 – 100th |
|---|---|---|---|---|
| Growth rate | (, 1.62) | [1.63, 3.89) | [3.90, 6.20) | [6.21, ) |
| TFP | (, 0.50) | [0.51, 0.71) | [0.72, 0.88) | [0.88, ) |
| Level | 0 | 1 | 2 | 3 |

The SITC 4-Code dataset has 9,009,106 observations on 205 countries (including autonomous territories) and 782 products from 1962 to 2017. After combining it with growth rate and TFP dataset, respectively, and taking only complete cases, the dimension of the final input dataset is (8486, 783) for growth rate prediction and (5447, 783) for TFP prediction.

## 3. Methods

*Baseline model*

The prediction models used in this study are Multi-layer Perceptron (MLP). In both growth rate prediction and TFP prediction, this study used three hidden layers MLP as a baseline model with the *ReLU* activation function for the three hidden layers and *softmax* function for the last output layer. After conducting a hyperparameter tuning process, in the growth model, this study used the batch size 128, and the number of nodes in each hidden layer as 32, 64, and 16. In

the TFP model, the batch size was 128, and the number of nodes in each hidden layers was 64, 64, and 16.

*Regularization model*

Because most countries usually have a comparative advantage on a handful of specific product groups, many of 782 entries have zero values. In order to prevent any resulting overfitting issues, L2 regularization and dropout option were alternately applied to the baseline model. As in the baseline model, the hyperparameter tuning process was conducted to find the best model by applying several different values of L2 lambda and dropout rates as well as different numbers of nodes in each hidden layer. The combinations of the hyperparameters which showed the best performance are as below. (The top 10 results of the hyperparameter combination are listed in Appendix 1.)

Table 2. The Result of Hyperparameter Tuning

| | | Growth | | | TFP | | |
|---|---|---|---|---|---|---|---|
| | | Layer 1 | Layer 2 | Layer 3 | Layer 1 | Layer 2 | Layer 3 |
| L2-regularization model | # of nodes in each layer | 32 | 32 | 16 | 64 | 32 | 32 |
| | Lambda | 0.05 | 0.1 | 0.01 | 0.05 | 0.05 | 0.05 |
| Dropout model | # of nodes in each layer | 32 | 32 | 16 | 64 | 32 | 16 |
| | Dropout rate | 0.8 | 0.2 | 0.4 | 0.8 | 0.6 | 0.4 |

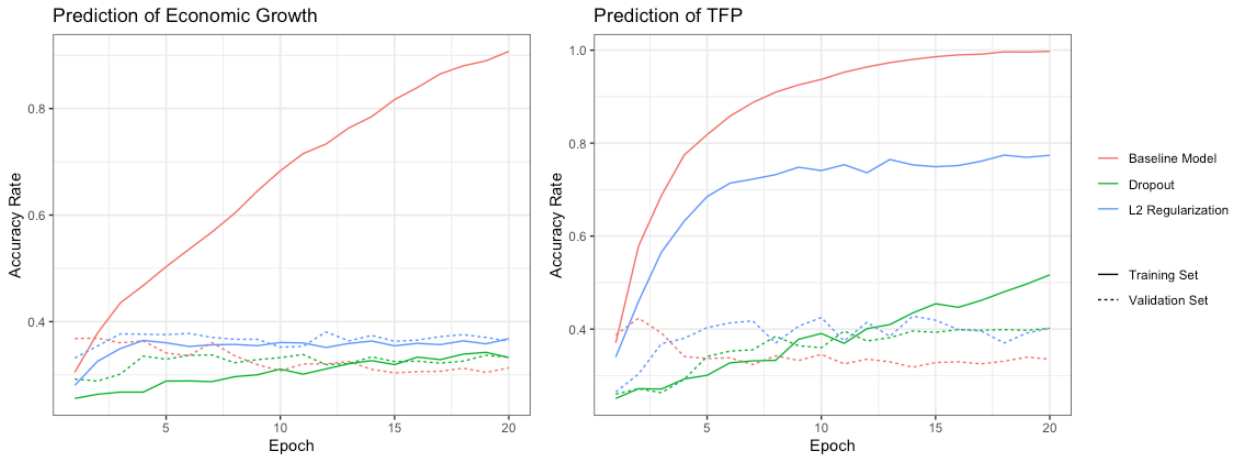
Comparing the two models, this study chose the best models for growth and TFG, respectively, fit it to the test dataset, and identified the final accuracy rate. The ratio of the test set is 20% of the whole dataset, and each training set has a validation set which is 20% of the training set.

Wooyong Jung
Net ID: wj710

## 4. Result

Using the validation dataset, the best model for predicting growth was the L2 regularization model, and that for predicting TFP was the dropout model as shown in Figure 1. As expected, the baseline model has an overfitting issue.

Figure 1. Prediction of Economic Growth and TFP




Using the best models, this study finally predicted the test set. The accuracy rates of prediction were 35.28% and 60.43% for economic growth and TFP, respectively, which tell us that RCA is a better predictor for estimating the latter than the former. However, it didn't predict both growth rate and TFP quite well, and there were several reasons for that. First of all, the models couldn't take advantage of the longitudinal data frame; instead, it collapsed the time information and pooled all observations at one-time point. Applying a recurrent neural network (RNN) method might improve the prediction accuracy. Secondly, because this study addressed the country-level data, there was a limitation in terms of the number of observations. Moreover,

small countries have many missing values in growth rate and TFP, the observations decreased further. Thus, it would be helpful if imputation was implemented before applying MLP.

## 5. Conclusion

This study tried to predict countries' economic performance measured by their economic growth rate and TFP by using their staple products as a predictor. Although the accuracy of the prediction was not high enough to say that RCA is a good predictor, this study found its potential. Letting it go through a better network such as RNN might improve its quality to a large extent, and securing more observations using other statistical methods might be also helpful.

Most of all, this study is meaningful in that it tested another potential dataset in addition to the global efforts applying machine leaning technics to the international development sector to fill up the gap of data availability between developed and developing countries. Because the trade dataset has been recorded by both exporters and importers, it has an advantage in terms of accuracy. If we can make good use of this advantage, it will redeem any possibilities of a lack of accuracy and consistency in developing countries' data., and, in turn, qualify RCA as a potential predictor for any other socioeconomic indicators.

## Reference

Balassa, B. 1986 "Comparative Advantage in Manufactured Goods: A Reappraisal." *The Review of Economics and Statistics* 68(2): 315-19.

Hausmann Ricardo, Jason Hwang, and Dani Rodrik. 2007. "What You Export Matters." *Journal of Economic Growth* 12: 1-25.

The Growth Lab at Harvard University. The Atlas of Economic Complexity. http://www.atlas.cid.harvard.edu.

## Appendix 1. The Result of Hyperparameter Tuning (Top 10 Results)

Table A1. Hyperparameter tuning for L2-regularization model of predicting the growth rate

| loss_train | accur_train | loss_val | accur_val | dense_units1 | dense_units2 | dense_units3 | L2_lambda1 | L2_lambda2 | L2_lambda3 | samples | batch_size | epochs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.3685 | 0.3612 | 1.3891 | 0.3814 | 32 | 32 | 16 | 0.05 | 0.1 | 0.01 | 5462 | 128 | 20 |
| 1.3475 | 0.3887 | 1.406 | 0.3799 | 64 | 32 | 32 | 0.01 | 0.05 | 0.1 | 5462 | 128 | 20 |
| 1.3617 | 0.3561 | 1.3963 | 0.3799 | 32 | 64 | 16 | 0.05 | 0.1 | 0.01 | 5462 | 128 | 20 |
| 1.3628 | 0.381 | 1.3859 | 0.3792 | 64 | 32 | 16 | 0.01 | 0.05 | 0.1 | 5462 | 128 | 20 |
| 1.3504 | 0.3903 | 1.3882 | 0.3785 | 64 | 64 | 32 | 0.05 | 0.01 | 0.05 | 5462 | 128 | 20 |
| 1.3466 | 0.3907 | 1.397 | 0.3777 | 64 | 32 | 32 | 0.01 | 0.1 | 0.05 | 5462 | 128 | 20 |
| 1.3617 | 0.3552 | 1.3897 | 0.3763 | 64 | 64 | 16 | 0.05 | 0.1 | 0.01 | 5462 | 128 | 20 |
| 1.354 | 0.3645 | 1.3911 | 0.3748 | 32 | 64 | 32 | 0.1 | 0.05 | 0.01 | 5462 | 128 | 20 |
| 1.3539 | 0.3916 | 1.4028 | 0.3734 | 32 | 64 | 16 | 0.01 | 0.05 | 0.1 | 5462 | 128 | 20 |
| 1.352 | 0.379 | 1.3862 | 0.3734 | 64 | 32 | 32 | 0.05 | 0.01 | 0.05 | 5462 | 128 | 20 |

Table A2. Hyperparameter tuning for Dropout model of predicting the growth rate

| loss_train | accur_train | loss_val | accur_val | dense_units1 | dense_units2 | dense_units3 | drop_out1 | drop_out2 | drop_out3 | samples | batch_size | epochs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.3243 | 0.3215 | 1.359 | 0.3602 | 32 | 32 | 16 | 0.8 | 0.2 | 0.4 | 5462 | 128 | 20 |
| 1.3543 | 0.3261 | 1.3566 | 0.3543 | 64 | 32 | 16 | 0.4 | 0.4 | 0.8 | 5462 | 128 | 20 |
| 1.2665 | 0.387 | 1.3482 | 0.3521 | 64 | 32 | 16 | 0.8 | 0.2 | 0.2 | 5462 | 128 | 20 |
| 1.2458 | 0.4259 | 1.3736 | 0.3507 | 32 | 32 | 16 | 0.4 | 0.8 | 0.2 | 5462 | 128 | 20 |
| 1.2628 | 0.3938 | 1.3651 | 0.3485 | 32 | 32 | 32 | 0.2 | 0.4 | 0.8 | 5462 | 128 | 20 |
| 1.2941 | 0.3477 | 1.338 | 0.3477 | 64 | 64 | 16 | 0.8 | 0.2 | 0.4 | 5462 | 128 | 20 |
| 1.345 | 0.3118 | 1.3572 | 0.347 | 64 | 64 | 16 | 0.4 | 0.4 | 0.8 | 5462 | 128 | 20 |
| 1.2965 | 0.3535 | 1.3492 | 0.3463 | 64 | 32 | 32 | 0.8 | 0.2 | 0.4 | 5462 | 128 | 20 |
| 1.1965 | 0.4431 | 1.3798 | 0.3455 | 64 | 64 | 32 | 0.4 | 0.2 | 0.8 | 5462 | 128 | 20 |
| 1.3164 | 0.3501 | 1.3541 | 0.3455 | 32 | 64 | 16 | 0.4 | 0.2 | 0.8 | 5462 | 128 | 20 |

Table A3. Hyperparameter tuning for L2-regularization model of predicting TFP

| loss_train | accur_train | loss_val | accur_val | dense_units1 | dense_units2 | dense_units3 | L2_lambda1 | L2_lambda2 | L2_lambda3 | samples | batch_size | epochs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.0318 | 0.759 | 1.743 | 0.4408 | 64 | 32 | 32 | 0.05 | 0.05 | 0.05 | 2768 | 128 | 20 |
| 0.9807 | 0.7717 | 1.777 | 0.4292 | 64 | 32 | 32 | 0.1 | 0.05 | 0.01 | 2768 | 128 | 20 |
| 1.1518 | 0.7088 | 1.6536 | 0.422 | 32 | 64 | 32 | 0.1 | 0.05 | 0.1 | 2768 | 128 | 20 |
| 1.1832 | 0.6926 | 1.603 | 0.422 | 32 | 32 | 32 | 0.1 | 0.05 | 0.1 | 2768 | 128 | 20 |
| 1.0807 | 0.7449 | 1.6579 | 0.422 | 32 | 64 | 32 | 0.05 | 0.1 | 0.05 | 2768 | 128 | 20 |
| 0.9405 | 0.8266 | 1.8883 | 0.422 | 64 | 64 | 16 | 0.01 | 0.1 | 0.05 | 2768 | 128 | 20 |
| 1.0209 | 0.7619 | 1.8115 | 0.422 | 64 | 64 | 32 | 0.05 | 0.05 | 0.05 | 2768 | 128 | 20 |
| 1.1396 | 0.7197 | 1.6323 | 0.4205 | 32 | 64 | 32 | 0.05 | 0.1 | 0.1 | 2768 | 128 | 20 |
| 0.9993 | 0.7645 | 1.8385 | 0.4205 | 32 | 32 | 32 | 0.1 | 0.01 | 0.05 | 2768 | 128 | 20 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.0259 | 0.7807 | 1.8951 | 0.4191 | 32 | 32 | 16 | 0.05 | 0.01 | 0.1 | 2768 | 128 | 20 |

Table A4. Hyperparameter tuning for Dropout model of predicting TFP

| loss_train | accur_train | loss_val | accur_val | dense_units1 | dense_units2 | dense_units3 | drop_out1 | drop_out2 | drop_out3 | samples | batch_size | epochs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.1524 | 0.4718 | 1.2774 | 0.4465 | 64 | 32 | 16 | 0.8 | 0.6 | 0.4 | 2768 | 128 | 20 |
| 1.1682 | 0.4382 | 1.238 | 0.4422 | 32 | 32 | 16 | 0.8 | 0.4 | 0.4 | 2768 | 128 | 20 |
| 1.1769 | 0.435 | 1.2966 | 0.435 | 64 | 64 | 32 | 0.8 | 0.6 | 0.6 | 2768 | 128 | 20 |
| 1.051 | 0.5116 | 1.258 | 0.4306 | 64 | 32 | 32 | 0.4 | 0.8 | 0.6 | 2768 | 128 | 20 |
| 0.8022 | 0.6579 | 1.573 | 0.4292 | 32 | 64 | 32 | 0.6 | 0.6 | 0.4 | 2768 | 128 | 20 |
| 1.2404 | 0.401 | 1.2871 | 0.4205 | 64 | 64 | 32 | 0.8 | 0.8 | 0.4 | 2768 | 128 | 20 |
| 1.0521 | 0.5098 | 1.2725 | 0.4191 | 64 | 64 | 16 | 0.8 | 0.6 | 0.4 | 2768 | 128 | 20 |
| 0.9455 | 0.5744 | 1.2928 | 0.4191 | 32 | 32 | 32 | 0.6 | 0.6 | 0.4 | 2768 | 128 | 20 |
| 0.9979 | 0.5466 | 1.253 | 0.4176 | 64 | 64 | 32 | 0.8 | 0.4 | 0.6 | 2768 | 128 | 20 |
| 1.2512 | 0.4249 | 1.3061 | 0.4176 | 64 | 32 | 16 | 0.8 | 0.4 | 0.6 | 2768 | 128 | 20 |

# Appendix 2. R code used in this study

```r
# Load libraries
library(haven)
library(ggplot2)
library(tidyverse)
library(keras)
library(tfruns)
```

```r
# Load dataset
## RCA data
load("Termpaper/dataset/country_sitcproduct4digit_year.RData")
## SITC product and country(location) code
pid <- read_dta("Termpaper/dataset/sitc_product.dta")
loc <- read_dta("Termpaper/dataset/location.dta")

# Merge pid and loc into main data frame
df <- merge(table, pid[,-5], by="product_id", all.x = TRUE)
df <- merge(df, loc[,c(1,2,3,4)], by="location_id", all.x = TRUE)

# Select variables needed for this study
df <- df %>%
  select(location_id, product_id, year, export_rca, sitc_product_name_short_en,
         location_code.x, location_name_short_en)

# Change column name
colnames(df) <- c("lid", "pid", "year", "rca", "product", "iso", "country")
```

```r
# Load gdp growth
gdp <- read_csv("Termpaper/dataset/gdp_growth.csv")

# Convert to long data
gdp <- gather(gdp, year, growth, `1961`:`2018`)

# Check quantile level
quantile(gdp$growth, na.rm = TRUE)

# Create growth level variable (1 to 4)
gdp %>%
  mutate(growth = ntile(growth, 4)) -> gdp

# Convert growth level 0 to 3
gdp <- gdp %>%
  mutate(growth = ifelse(growth == 1, 0,
                         ifelse(growth == 2, 1,
                                ifelse(growth == 3, 2,
                                       ifelse(growth == 4, 3, NA)))))
```

```r
# Load Penn World Table data
pwt <- read_csv("Termpaper/dataset/penntable.csv")

# Choose variables needed
pwt <- pwt %>%
  select(iso, country, year, ctfp)

# Convert to long data
pwt %>%
  mutate(tfp = ntile(ctfp, 5)) -> pwt

# Check quantile level
quantile(pwt$ctfp, na.rm = TRUE)
pwt <- pwt[,-4]

# Convert TFP level 0 to 3
pwt <- pwt %>%
  mutate(tfp = ifelse(tfp == 1, 0,
                      ifelse(tfp == 2, 1,
                             ifelse(tfp == 3, 2,
                                    ifelse(tfp == 4, 3, NA)))))
```

```r
# Merge growth into df and make new dataframe (df.growth)
df_growth <- merge(df, gdp, by = c("iso", "year"), all.x = TRUE)

# Remain only complete cases and select variables needed
df_growth <- df_growth %>%
  drop_na() %>%
  select(iso, year, pid, rca, growth)

# Convert to wide format (products in column) and Change NA to 0
## NA means that the country has no RCA of the product
df_growth <- df_growth %>%
  spread(pid, rca) %>%
  mutate_at(vars(3:785), funs(ifelse(is.na(.), 0, .)))

# Convert RCA as a binary
df_growth <- df_growth %>%
  mutate_at(vars(4:785), funs(ifelse(.>1, 1, 0)))

# Merge TFP into df and make new dataframe (df_tfp)
df_tfp <- merge(df, pwt, by = c("iso", "year"), all.x = TRUE)

# Remain only complete cases and select variables needed
df_tfp <- df_tfp %>%
  drop_na() %>%
  select(iso, year, pid, rca, tfp)

# Convert  to wide (products in column) and Change NA to 0
## NA means that the country has no RCA of the product
df_tfp <- df_tfp %>%
  spread(pid, rca) %>%
  mutate_at(vars(3:785), funs(ifelse(is.na(.), 0, .)))
```

```r
# Convert RCA as a binary
df_tfp <- df_tfp %>%
  mutate_at(vars(4:785), funs(ifelse(.>1, 1, 0)))
```

```r
# Convert df.growth into a matrix
df_growth <- df_growth[,-c(1,2)]
mat_growth <- as.matrix(df_growth)
dimnames(mat_growth) <- NULL

# Seperate into train and test set
set.seed(123)
idx_growth <- sample(2, nrow(mat_growth), replace = TRUE, prob = c(0.8, 0.2))
growth_x_train <- mat_growth[idx_growth == 1, 2:783]
growth_x_test <- mat_growth[idx_growth == 2, 2:783]

growth_y_test_actual <- mat_growth[idx_growth == 2, 1]

growth_y_train <- to_categorical(mat_growth[idx_growth == 1, 1])
growth_y_test <- to_categorical(mat_growth[idx_growth == 2, 1])

cbind(growth_y_test_actual[1:10], growth_y_test[1:10,])
```

```r
# Hyperparameter tuning
run_bm_growth <- tuning_run("tuning1.R",
                   flags = list(dense_units1 = c(32, 64),
                                dense_units2 = c(32, 64),
                                dense_units3 = c(16, 32),
                                batch_size = c(32, 64, 128)))
write.csv(run_bm_growth, "bm_growth.csv")

# Best hyperparameter values
head(runs)

# Create a baseline model
bm_growth <- keras_model_sequential()
bm_growth %>%
  layer_dense(units = 32, activation = "relu", input_shape = 782) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 4, activation = "softmax")
summary(bm_growth)

# Compile the model
bm_growth %>%
  compile(loss = "categorical_crossentropy",
          optimizer = "adam",
          metrics = c("accuracy"))

# Fit the data
bm_growth_history <- bm_growth %>%
  fit(growth_x_train, growth_y_train, epoch = 20, batch_size = 128,
      validation_split = 0.2, verbose = 2)
```

```r
# Hyperparameter tuning
runs_l2_growth <- tuning_run("tuning2.R",
                   flags = list(dense_units1 = c(32, 64),
                                dense_units2 = c(32, 64),
                                dense_units3 = c(16, 32),
                                l2_lambda1 = c(0.01, 0.05, 0.1),
                                l2_lambda2 = c(0.01, 0.05, 0.1),
                                l2_lambda3 = c(0.01, 0.05, 0.1)))
write.csv(runs_l2_growth, "l2_growth.csv")

# Create a baseline model
l2_growth <- keras_model_sequential()
l2_growth %>%
  layer_dense(units = 32, activation = "relu", input_shape = 782,
              kernel_regularizer = regularizer_l2(0.05)) %>%
  layer_dense(units = 32, activation = "relu",
              kernel_regularizer = regularizer_l2(0.1)) %>%
  layer_dense(units = 16, activation = "relu",
              kernel_regularizer = regularizer_l2(0.01)) %>%
  layer_dense(units = 4, activation = "softmax")

# Compile the model
l2_growth %>%
  compile(loss = "categorical_crossentropy",
          optimizer = "adam",
          metrics = c("accuracy"))

# Fit the data
l2_growth_history <- l2_growth %>%
  fit(growth_x_train, growth_y_train, epoch = 20, batch_size = 128,
      validation_split = 0.2, verbose = 2)
```

```r
# Hyperparameter tuning
runs_dp_growth <- tuning_run("tuning3.R",
                   flags = list(dense_units1 = c(32, 64),
                                dense_units2 = c(32, 64),
                                dense_units3 = c(16, 32),
                                dropout1 = c(0.2, 0.4, 0.8),
                                dropout2 = c(0.2, 0.4, 0.8),
                                dropout3 = c(0.2, 0.4, 0.8)))
write.csv(runs_dp_growth, "dp_growth.csv")

# Create a dropout model
dp_growth <- keras_model_sequential()
dp_growth %>%
  layer_dense(units = 32, activation = "relu", input_shape = 782) %>%
  layer_dropout(0.8) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dropout(0.2) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dropout(0.4) %>%
  layer_dense(units = 4, activation = "softmax")

# Compile the model
```

```r
dp_growth %>%
  compile(loss = "categorical_crossentropy",
          optimizer = "adam",
          metrics = c("accuracy"))

# Fit the data
dp_growth_history <- dp_growth %>%
  fit(growth_x_train, growth_y_train, epoch = 20, batch_size = 128,
      validation_split = 0.2, verbose = 2)
```

```r
# Make a data frame including accuracy of training and validation set
compare_growth <- data.frame(
  baseline_train = bm_growth_history$metrics$accuracy,
  baseline_val = bm_growth_history$metrics$val_accuracy,
  l2_train = l2_growth_history$metrics$accuracy,
  l2_val = l2_growth_history$metrics$val_accuracy,
  dp_train = dp_growth_history$metrics$accuracy,
  dp_val = dp_growth_history$metrics$val_accuracy) %>%
  rownames_to_column() %>%
  mutate(rowname = as.integer(rowname)) %>%
  gather(key = "model", value = "accuracy", -rowname)

# Create index variable indicating each model and dataset type
compare_growth$type[grep("baseline", compare_growth$model)] <- "Baseline Model"
compare_growth$type[grep("l2", compare_growth$model)] <- "L2 Regularization"
compare_growth$type[grep("dp", compare_growth$model)] <- "Dropout"

compare_growth$type2 <- NA
compare_growth$type2[grep("train", compare_growth$model)] <- "Training Set"
compare_growth$type2[grep("val", compare_growth$model)] <- "Validation Set"

# Plot
compare_growth %>%
  ggplot(aes(x = rowname, y = accuracy, color = type, group = model, linetype = type2)) +
  geom_line() +
  theme_bw() +
  labs(x = "Epoch", y = "Accuracy Rate", title = "Prediction of Economic Growth") +
  theme(legend.title = element_blank()) -> plot_growth
plot_growth
```

```r
# Apply L2 model (best model) to test set
metrics_growth <- l2_growth %>%
  evaluate(growth_x_test, growth_y_test)
metrics_growth
```

```r
# Convert df_tfp into a matrix
df_tfp <- df_tfp[,-c(1,2)]
mat_tfp <- as.matrix(df_tfp)
dimnames(mat_tfp) = NULL

# Seperate into train and test set
set.seed(123)
idx_tfp <- sample(2, nrow(mat_tfp), replace = TRUE, prob = c(0.8, 0.2))
tfp_x_train <- mat_tfp[idx_tfp == 1, 2:783]
```

```r
tfp_x_test <- mat_tfp[idx_tfp == 2, 2:783]

tfp_y_test_actual <- mat_tfp[idx_tfp == 2, 1]

tfp_y_train <- to_categorical(mat_tfp[idx_tfp == 1, 1])
tfp_y_test <- to_categorical(mat_tfp[idx_tfp == 2, 1])

cbind(tfp_y_test_actual[1:10], tfp_y_test[1:10,])
```

```r
# Hyperparameter tuning
runs_bm_tfp <- tuning_run("tuning4.R",
                  flags = list(dense_units1 = c(32, 64),
                               dense_units2 = c(32, 64),
                               dense_units3 = c(16, 32),
                               batch_size = c(32, 64, 128)))
write.csv(runs_bm_tfp, "bm_tfp.csv")

# Best hyperparameter values
head(runs)

# Create a baseline model
bm_tfp <- keras_model_sequential()
bm_tfp %>%
  layer_dense(units = 64, activation = "relu", input_shape = 782) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 4, activation = "softmax")
summary(bm_tfp)

# Compile the model
bm_tfp %>%
  compile(loss = "categorical_crossentropy",
          optimizer = "adam",
          metrics = c("accuracy"))

# Fit the data
bm_tfp_history <- bm_tfp %>%
  fit(tfp_x_train, tfp_y_train, epoch = 20, batch_size = 128,
      validation_split = 0.2, verbose = 2)
```

```r
# Hyperparameter tuning
runs_l2_tfp <- tuning_run("tuning5.R",
                  flags = list(dense_units1 = c(32, 64),
                               dense_units2 = c(32, 64),
                               dense_units3 = c(16, 32),
                               l2_lambda1 = c(0.01, 0.05, 0.1),
                               l2_lambda2 = c(0.01, 0.05, 0.1),
                               l2_lambda3 = c(0.01, 0.05, 0.1)))
write.csv(runs_l2_tfp, "l2_tfp.csv")

# Create a baseline model
l2_tfp <- keras_model_sequential()
l2_tfp %>%
  layer_dense(units = 64, activation = "relu", input_shape = 782,
```

```
              kernel_regularizer = regularizer_l2(0.05)) %>%
  layer_dense(units = 32, activation = "relu",
              kernel_regularizer = regularizer_l2(0.05)) %>%
  layer_dense(units = 32, activation = "relu",
              kernel_regularizer = regularizer_l2(0.05)) %>%
  layer_dense(units = 4, activation = "softmax")
summary(l2_tfp)

# Compile the model
l2_tfp %>%
  compile(loss = "categorical_crossentropy",
          optimizer = "adam",
          metrics = c("accuracy"))

# Fit the data
l2_tfp_history <- l2_tfp %>%
  fit(tfp_x_train, tfp_y_train, epoch = 20, batch_size = 128,
      validation_split = 0.2, verbose = 2)
```

```
# Hyperparameter tuning
runs_dp_tfp <- tuning_run("tuning6.R",
                  flags = list(dense_units1 = c(32, 64),
                               dense_units2 = c(32, 64),
                               dense_units3 = c(16, 32),
                               dropout1 = c(0.4, 0.6, 0.8),
                               dropout2 = c(0.4, 0.6, 0.8),
                               dropout3 = c(0.4, 0.6, 0.8)))
write.csv(runs_dp_tfp, "dp_tfp.csv")

# Create a dropout model
dp_tfp <- keras_model_sequential()
dp_tfp %>%
  layer_dense(units = 64, activation = "relu", input_shape = 782) %>%
  layer_dropout(0.8) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dropout(0.6) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dropout(0.4) %>%
  layer_dense(units = 4, activation = "softmax")
summary(dp_tfp)

# Compile the model
dp_tfp %>%
  compile(loss = "categorical_crossentropy",
          optimizer = "adam",
          metrics = c("accuracy"))

# Fit the data
dp_tfp_history <- dp_tfp %>%
  fit(tfp_x_train, tfp_y_train, epoch = 20, batch_size = 128,
      validation_split = 0.2, verbose = 2)
```

```
# Make a data frame including accuracy of training and validation set
compare_tfp <- data.frame(
```

```
  baseline_train = bm_tfp_history$metrics$accuracy,
  baseline_val = bm_tfp_history$metrics$val_accuracy,
  l2_train = l2_tfp_history$metrics$accuracy,
  l2_val = l2_tfp_history$metrics$val_accuracy,
  dp_train = dp_tfp_history$metrics$accuracy,
  dp_val = dp_tfp_history$metrics$val_accuracy) %>%
  rownames_to_column() %>%
  mutate(rowname = as.integer(rowname)) %>%
  gather(key = "model", value = "accuracy", -rowname)

# Create index variable indicating each model and dataset type
compare_tfp$type[grep("baseline", compare_tfp$model)] <- "Baseline Model"
compare_tfp$type[grep("l2", compare_tfp$model)] <- "L2 Regularization"
compare_tfp$type[grep("dp", compare_tfp$model)] <- "Dropout"

compare_tfp$type2 <- NA
compare_tfp$type2[grep("train", compare_tfp$model)] <- "Training Set"
compare_tfp$type2[grep("val", compare_tfp$model)] <- "Validation Set"

# Plot
compare_tfp %>%
  ggplot(aes(x = rowname, y = accuracy, color = type, group = model, linetype = type2)) +
  geom_line() +
  theme_bw() +
  labs(x = "Epoch", y = "Accuracy Rate", title = "Prediction of TFP") +
  theme(legend.title = element_blank()) -> plot_tfp
plot_tfp
```

```
# Apply Dropout model (best model) to test set
metrics_tfp <- dp_tfp %>%
  evaluate(tfp_x_test, tfp_y_test)
metrics_tfp
```