Kaggle 3

Nicholas Scholl

2023-11-02

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

```
#install.packages("keras")
#devtools::install_github("rstudio/keras")
#install.packages("keras")
#keras:: install_keras()
```

Here the keras package is installed and later called to utilize the full capabilities. The previous code lines are commented out as we do not need them

```
library(tensorflow)
library(keras)
mnist <- dataset_mnist()
#x_train <- mnist$train$x
#y_train <- mnist$train$y
#x_test <- mnist$test$x
#y_test <- mnist$test$y
```

During testing, the data set from the keras package was used, this came in as 1X28X28 array (an image of the numbers) and could be converted into a matrix in a later chunk. The data is similar to the one provided in kaggle, both are the same size. To provide authenticitiy, we will use the data set that was provided in kaggle.

```
x <- as.matrix(read.csv("mnist_train.csv"))
y <- as.vector(read.csv("mnist_train_targets.csv")$x)
x_test <- as.matrix(read.csv("mnist_test.csv"))
```

This is loading in the data from the kaggle.

```
x <- t(x)
x_test <- t(x_test)
x <- apply(x, 2, function(y) y / 255)
x_test <- apply(x_test, 2, function(y) y / 255)
```

Transforming the data that was provided in kaggle. Here the entries are divided by 255. This is because each pixel of the data sets have a brightness number. This is anywhere between 0-255, with 255 being the brightest. The data transforms into percentages of brightness for ease of use in the functions below. So now, all entries will be anywhere from 0-1 for brightness of that pixel.

```
#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
```

In this chunk, The data is transformed from keras from an array into a matrix. To use it for our purposes, the data is divide each entry by 255 to bring a brightness percentage as was done before.

```
#y_train <- to_categorical(y_train, 10)
#y_test <- to_categorical(y_test, 10)
y <- to_categorical(y, 10)
```

Making the Y sets categorical for ease of use. The data is made into categories to make each prediction into buckets. Each prediction is put into the category of what the neural net believes that hand written number to be. Since the data from keras will not be used, it can be commented out.

```
library(readr)
TestingOfNN <- as.data.frame(read_csv("TestingOfNN.csv", show_col_types = FALSE))
```

Loading in the testing data set that was recorded during experimentation of hyperparameters.

```
#activationFN1 <- TestingOfNN %>%
  #group_by(TestingOfNN$`Activation function 1`) %>%

#ggplot2::ggplot(data = TestingOfNN, aes(x = `Activation function 1`, y = `Accuracy`, fill = `Activatio
  #geom_col(stat = "Identity", position = "dodge") +
  #scale_y_continuous(name = 'Accuracy', limits = c(95,100))
print(TestingOfNN)
```

```
##    Layers Hidden layer 1 Hidden layer 2 Hidden layer 3 Activation function 1
## 1       3            256            128             64                  Relu
## 2       2            128             64             NA               Sigmoid
## 3       2            128             64             NA                  Tanh
## 4       2            128             64             NA               Softmax
## 5       3            256            256            256                  Relu
## 6       3             64             64             64               Sigmoid
## 7       3            256            128             64                  Relu
## 8       3            256            128             64                  Relu
## 9       3            256            128             64                  Tanh
## 10      3            256            128             64                  Relu
##    Activation Function 2 Activation 3 Drop out rate Activation final Accuracy
## 1                   Relu         Relu          0.25          Softmax   99.500
## 2                Sigmoid         <NA>          0.25          Softmax   99.300
## 3                   Tanh         <NA>          0.25         Softplus   95.160
## 4                Softmax         <NA>          0.25         Softplus   95.160
## 5                   Relu         Relu          0.25          Softmax   98.766
```

```
## 6                Sigmoid      Sigmoid          0.25          Softmax   98.600
## 7                Sigmoid         Relu          0.25          Softmax   99.600
## 8                Sigmoid         Relu          0.40          Softmax   99.633
## 9                Sigmoid         Tanh          0.40          Softmax   99.366
## 10              Softplus         Relu          0.40          Softmax   99.300
```

```
TestingOfNN[order(TestingOfNN$Accuracy),]
```

```
##    Layers Hidden layer 1 Hidden layer 2 Hidden layer 3 Activation function 1
## 3       2            128             64             NA                   Tanh
## 4       2            128             64             NA                Softmax
## 6       3             64             64             64                Sigmoid
## 5       3            256            256            256                   Relu
## 2       2            128             64             NA                Sigmoid
## 10      3            256            128             64                   Relu
## 9       3            256            128             64                   Tanh
## 1       3            256            128             64                   Relu
## 7       3            256            128             64                   Relu
## 8       3            256            128             64                   Relu
##    Activation Function 2 Activation 3 Drop out rate Activation final Accuracy
## 3                   Tanh         <NA>          0.25         Softplus   95.160
## 4                Softmax         <NA>          0.25         Softplus   95.160
## 6                Sigmoid      Sigmoid          0.25          Softmax   98.600
## 5                   Relu         Relu          0.25          Softmax   98.766
## 2                Sigmoid         <NA>          0.25          Softmax   99.300
## 10              Softplus         Relu          0.40          Softmax   99.300
## 9                Sigmoid         Tanh          0.40          Softmax   99.366
## 1                   Relu         Relu          0.25          Softmax   99.500
## 7                Sigmoid         Relu          0.25          Softmax   99.600
## 8                Sigmoid         Relu          0.40          Softmax   99.633
```

From the results of this experimentation. It is shown that 3 hidden layers, 256 neurons for the first layer with relu activation, 128 for the second layer with sigmoid and 64 for the last hidden layer with the final function being softmax, produced the best results from testing. Thus, the final product will contain these hyperparamters.

```
model <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu", input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = "sigmoid") %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 10, activation = "softmax")
```

This chunk is where the neural net is built. First, the keras model sequential function is called to specify that a linear stack of layers will be used for the keras model. Next, is to build the first layer. Based on testing, relu in the first layer with 256 neurons performed the best in terms of accuracy. So, units declared are 256, and the chunk adds another argument for input shape, this is the input neuron layers. Earlier, the data set was provided as 60,000 entries with 784 attributes, each attribute represents 1 pixel in the image. Because of that, 784 input neurons are declared. The next thing to speak about would be the drop out rate. Drop out rate is randomly zeroing out a fraction of neurons to prevent overfitting. This is to prevent neurons from having dependencies on some other neurons and cannot generalize the model. These steps are

repeated to include another layer for 128 with sigmoid. Another layer for 64 using relu and lastly, a softmax activation function to send to the output neurons to send to a 0-9 category of outputs. This code was from Maria Grycuk from her guide:https://appsilon.com/r-keras-mnist/. This code chunk was transformed into my current model but the base code is from her.

```
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_adam(),
  metrics = c("accuracy")
)
```

This chunk is to specify in our model certain aspects such as loss, optimizers and a metric. Categorical crossentropy is chosen as the loss function since the problem is to set certain images to certain categories or buckets of what the model believes is the hand written digit. In this, the final output will be a vector of 9 zeros and a single 1 for what the model believes is that number. Next, optimizer adam is chosen for the model because of its ability take into consideration the exponentially weighted average of the gradient descent and accelerates it. The algorithm runs wonderfully in our example and is often used for large data sets for machine learning. Lastly, a graph is created with the metrics function. That is set to accuracy as this project is aiming to predict hand written digits and this is determined by accuracy of the model. This code was from Maria Grycuk from her guide:https://appsilon.com/r-keras-mnist/.

```
history <- model %>%
  fit(x, y, epochs = 50, batch_size = 128, validation_split = 0.15)
```

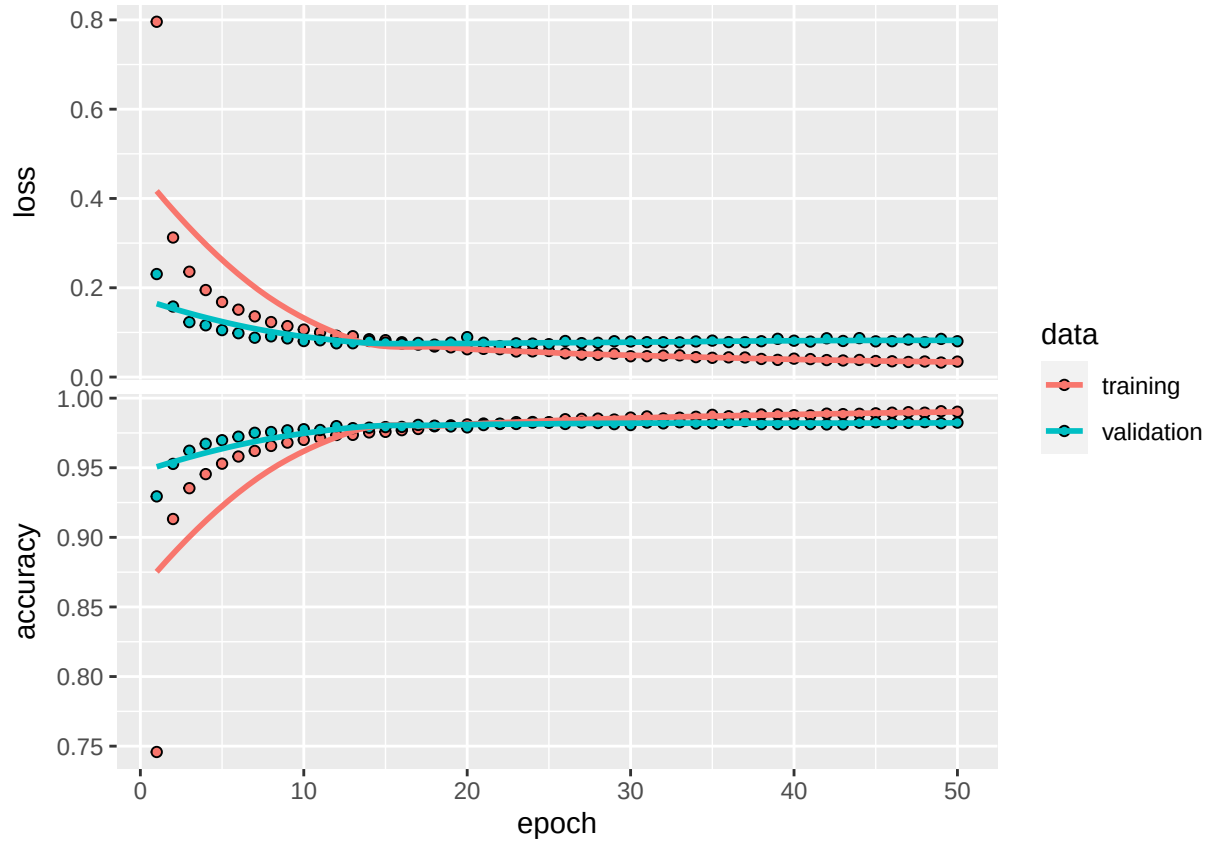
```
## Epoch 1/50
## 399/399 - 3s - loss: 0.7958 - accuracy: 0.7458 - val_loss: 0.2307 - val_accuracy: 0.9294 - 3s/epoch -
## Epoch 2/50
## 399/399 - 2s - loss: 0.3125 - accuracy: 0.9132 - val_loss: 0.1580 - val_accuracy: 0.9528 - 2s/epoch -
## Epoch 3/50
## 399/399 - 2s - loss: 0.2359 - accuracy: 0.9353 - val_loss: 0.1230 - val_accuracy: 0.9622 - 2s/epoch -
## Epoch 4/50
## 399/399 - 2s - loss: 0.1949 - accuracy: 0.9454 - val_loss: 0.1160 - val_accuracy: 0.9672 - 2s/epoch -
## Epoch 5/50
## 399/399 - 2s - loss: 0.1682 - accuracy: 0.9529 - val_loss: 0.1052 - val_accuracy: 0.9698 - 2s/epoch -
## Epoch 6/50
## 399/399 - 2s - loss: 0.1509 - accuracy: 0.9581 - val_loss: 0.0983 - val_accuracy: 0.9723 - 2s/epoch -
## Epoch 7/50
## 399/399 - 2s - loss: 0.1360 - accuracy: 0.9621 - val_loss: 0.0881 - val_accuracy: 0.9751 - 2s/epoch -
## Epoch 8/50
## 399/399 - 2s - loss: 0.1231 - accuracy: 0.9657 - val_loss: 0.0912 - val_accuracy: 0.9757 - 2s/epoch -
## Epoch 9/50
## 399/399 - 2s - loss: 0.1141 - accuracy: 0.9681 - val_loss: 0.0866 - val_accuracy: 0.9769 - 2s/epoch -
## Epoch 10/50
## 399/399 - 2s - loss: 0.1064 - accuracy: 0.9700 - val_loss: 0.0806 - val_accuracy: 0.9777 - 2s/epoch -
## Epoch 11/50
## 399/399 - 2s - loss: 0.1002 - accuracy: 0.9712 - val_loss: 0.0823 - val_accuracy: 0.9771 - 2s/epoch -
## Epoch 12/50
## 399/399 - 2s - loss: 0.0926 - accuracy: 0.9735 - val_loss: 0.0756 - val_accuracy: 0.9799 - 2s/epoch -
## Epoch 13/50
## 399/399 - 2s - loss: 0.0915 - accuracy: 0.9736 - val_loss: 0.0757 - val_accuracy: 0.9784 - 2s/epoch -
## Epoch 14/50
## 399/399 - 2s - loss: 0.0847 - accuracy: 0.9754 - val_loss: 0.0808 - val_accuracy: 0.9788 - 2s/epoch -
## Epoch 15/50
```

```
## 399/399 - 3s - loss: 0.0826 - accuracy: 0.9758 - val_loss: 0.0772 - val_accuracy: 0.9792 - 3s/epoch -
## Epoch 16/50
## 399/399 - 3s - loss: 0.0780 - accuracy: 0.9770 - val_loss: 0.0749 - val_accuracy: 0.9793 - 3s/epoch -
## Epoch 17/50
## 399/399 - 2s - loss: 0.0726 - accuracy: 0.9779 - val_loss: 0.0764 - val_accuracy: 0.9808 - 2s/epoch -
## Epoch 18/50
## 399/399 - 2s - loss: 0.0684 - accuracy: 0.9798 - val_loss: 0.0726 - val_accuracy: 0.9802 - 2s/epoch -
## Epoch 19/50
## 399/399 - 3s - loss: 0.0666 - accuracy: 0.9805 - val_loss: 0.0772 - val_accuracy: 0.9797 - 3s/epoch -
## Epoch 20/50
## 399/399 - 2s - loss: 0.0622 - accuracy: 0.9811 - val_loss: 0.0895 - val_accuracy: 0.9790 - 2s/epoch -
## Epoch 21/50
## 399/399 - 3s - loss: 0.0632 - accuracy: 0.9817 - val_loss: 0.0769 - val_accuracy: 0.9807 - 3s/epoch -
## Epoch 22/50
## 399/399 - 2s - loss: 0.0625 - accuracy: 0.9814 - val_loss: 0.0693 - val_accuracy: 0.9817 - 2s/epoch -
## Epoch 23/50
## 399/399 - 2s - loss: 0.0573 - accuracy: 0.9828 - val_loss: 0.0756 - val_accuracy: 0.9814 - 2s/epoch -
## Epoch 24/50
## 399/399 - 3s - loss: 0.0575 - accuracy: 0.9828 - val_loss: 0.0750 - val_accuracy: 0.9824 - 3s/epoch -
## Epoch 25/50
## 399/399 - 3s - loss: 0.0583 - accuracy: 0.9827 - val_loss: 0.0736 - val_accuracy: 0.9823 - 3s/epoch -
## Epoch 26/50
## 399/399 - 3s - loss: 0.0532 - accuracy: 0.9849 - val_loss: 0.0807 - val_accuracy: 0.9814 - 3s/epoch -
## Epoch 27/50
## 399/399 - 2s - loss: 0.0502 - accuracy: 0.9851 - val_loss: 0.0760 - val_accuracy: 0.9824 - 2s/epoch -
## Epoch 28/50
## 399/399 - 3s - loss: 0.0497 - accuracy: 0.9853 - val_loss: 0.0767 - val_accuracy: 0.9821 - 3s/epoch -
## Epoch 29/50
## 399/399 - 3s - loss: 0.0526 - accuracy: 0.9846 - val_loss: 0.0803 - val_accuracy: 0.9813 - 3s/epoch -
## Epoch 30/50
## 399/399 - 2s - loss: 0.0466 - accuracy: 0.9860 - val_loss: 0.0797 - val_accuracy: 0.9807 - 2s/epoch -
## Epoch 31/50
## 399/399 - 2s - loss: 0.0468 - accuracy: 0.9869 - val_loss: 0.0783 - val_accuracy: 0.9824 - 2s/epoch -
## Epoch 32/50
## 399/399 - 3s - loss: 0.0481 - accuracy: 0.9854 - val_loss: 0.0780 - val_accuracy: 0.9819 - 3s/epoch -
## Epoch 33/50
## 399/399 - 2s - loss: 0.0486 - accuracy: 0.9859 - val_loss: 0.0778 - val_accuracy: 0.9827 - 2s/epoch -
## Epoch 34/50
## 399/399 - 2s - loss: 0.0447 - accuracy: 0.9866 - val_loss: 0.0798 - val_accuracy: 0.9818 - 2s/epoch -
## Epoch 35/50
## 399/399 - 3s - loss: 0.0432 - accuracy: 0.9880 - val_loss: 0.0817 - val_accuracy: 0.9820 - 3s/epoch -
## Epoch 36/50
## 399/399 - 3s - loss: 0.0441 - accuracy: 0.9870 - val_loss: 0.0783 - val_accuracy: 0.9824 - 3s/epoch -
## Epoch 37/50
## 399/399 - 3s - loss: 0.0439 - accuracy: 0.9870 - val_loss: 0.0780 - val_accuracy: 0.9834 - 3s/epoch -
## Epoch 38/50
## 399/399 - 3s - loss: 0.0406 - accuracy: 0.9883 - val_loss: 0.0803 - val_accuracy: 0.9813 - 3s/epoch -
## Epoch 39/50
## 399/399 - 2s - loss: 0.0385 - accuracy: 0.9883 - val_loss: 0.0856 - val_accuracy: 0.9813 - 2s/epoch -
## Epoch 40/50
## 399/399 - 3s - loss: 0.0412 - accuracy: 0.9878 - val_loss: 0.0814 - val_accuracy: 0.9817 - 3s/epoch -
## Epoch 41/50
## 399/399 - 3s - loss: 0.0403 - accuracy: 0.9876 - val_loss: 0.0795 - val_accuracy: 0.9813 - 3s/epoch -
## Epoch 42/50
```

```
## 399/399 - 3s - loss: 0.0380 - accuracy: 0.9889 - val_loss: 0.0868 - val_accuracy: 0.9811 - 3s/epoch -
## Epoch 43/50
## 399/399 - 3s - loss: 0.0371 - accuracy: 0.9885 - val_loss: 0.0809 - val_accuracy: 0.9811 - 3s/epoch -
## Epoch 44/50
## 399/399 - 3s - loss: 0.0383 - accuracy: 0.9887 - val_loss: 0.0872 - val_accuracy: 0.9823 - 3s/epoch -
## Epoch 45/50
## 399/399 - 3s - loss: 0.0360 - accuracy: 0.9891 - val_loss: 0.0800 - val_accuracy: 0.9827 - 3s/epoch -
## Epoch 46/50
## 399/399 - 3s - loss: 0.0353 - accuracy: 0.9895 - val_loss: 0.0803 - val_accuracy: 0.9821 - 3s/epoch -
## Epoch 47/50
## 399/399 - 3s - loss: 0.0338 - accuracy: 0.9899 - val_loss: 0.0840 - val_accuracy: 0.9822 - 3s/epoch -
## Epoch 48/50
## 399/399 - 3s - loss: 0.0348 - accuracy: 0.9895 - val_loss: 0.0781 - val_accuracy: 0.9828 - 3s/epoch -
## Epoch 49/50
## 399/399 - 3s - loss: 0.0325 - accuracy: 0.9906 - val_loss: 0.0855 - val_accuracy: 0.9820 - 3s/epoch -
## Epoch 50/50
## 399/399 - 3s - loss: 0.0346 - accuracy: 0.9902 - val_loss: 0.0802 - val_accuracy: 0.9824 - 3s/epoch -
```

This chunk is where the data is actually fit within the algorithm. The data chunk specifys the epoch as 50 to save on runtime and not overstrain any machine that might run this code. Epochs is the number of times a training data set will take around the algorithm. Batch size is the number of samples that will be passed within the model before the model updates itself with backpropagation and using optimizers. This code was from Maria Grycuk from her guide:https://appsilon.com/r-keras-mnist/.

```
plot(history)
```



Here is the plot of fitting the model with the training data. Validation is the value of the loss function with

our cross validation data (15% of the data). This is also true for the accuracy and using the validation split data to cross validate. Ideally, accuracy should be maximized and loss should be minimized. Using gradient descent, the weights and biases are changed during backpropagation to produce a model that minimizes the loss and maximizes the accuracy.

```
x_test <- as.matrix(read.csv("mnist_test.csv"))
x_test <- t(x_test)
#x_test <- (x_test), c(784, ncol(x_test)))
x_test <- x_test / 255
#model %>% evaluate(x_test, y_test)
```

The test data set that was provided in kaggle is loaded in here. Through trial and error, it was discovered that the current state of the data could not be fully used in the algorithm. A simple transpose was the key to be able to plug it into the function itself. It is normalized by dividing all entries by 255.

```
predictions <- model %>%
  predict(x_test) %>%
  k_argmax()
```

```
## 313/313 - 1s - 528ms/epoch - 2ms/step
```

```
predictions$numpy()
```

```
##     [1] 3 4 1 1 4 1 8 9 1 3 4 0 0 7 7 1 0 8 2 0 3 5 3 2 3 4 2 0 1 9 1 8 6 1 9 2
##    [37] 2 5 5 9 2 7 6 3 1 0 8 8 9 2 7 2 9 2 8 3 7 7 4 5 9 6 4 8 1 6 5 0 7 5 6 9
##    [73] 0 9 1 4 9 2 3 5 2 6 5 7 8 8 6 0 4 2 8 1 0 1 0 2 3 4 1 2 7 9 3 8 4 8 0 6
##   [109] 7 5 4 2 4 3 1 1 3 4 4 6 7 0 5 6 3 4 4 0 3 0 7 1 6 3 2 1 3 1 8 2 4 2 2 8
##   [145] 6 5 5 1 3 9 3 8 7 8 3 7 7 3 1 7 1 5 5 8 8 4 1 6 9 7 3 2 7 3 8 4 1 6 0 7
##   [181] 3 2 7 9 1 6 9 2 4 5 5 5 8 3 2 5 4 0 5 5 9 5 8 7 5 6 8 1 3 3 0 8 5 4 8 3
##   [217] 0 7 4 9 7 9 7 6 1 7 1 1 5 6 7 8 8 2 7 2 6 8 8 2 1 1 1 6 0 0 5 7 7 0 4 8
##   [253] 8 3 3 7 7 6 6 3 7 0 4 6 5 3 3 4 5 1 3 7 7 5 9 7 6 7 5 3 7 9 5 2 1 7 9 8
##   [289] 1 0 7 6 4 2 6 7 2 0 2 7 5 4 7 1 4 6 4 8 0 1 5 2 4 0 5 4 4 9 7 3 4 7 9 1
##   [325] 3 1 6 9 9 3 8 6 8 9 6 9 0 7 2 2 3 3 9 9 6 1 2 7 4 9 8 8 2 7 8 3 6 3 1 4
##   [361] 4 9 4 9 2 0 5 3 8 5 5 0 0 9 7 9 9 8 7 3 3 8 4 0 6 7 9 1 5 3 6 5 3 3 8 8
##   [397] 0 1 8 4 7 4 1 7 6 3 7 6 0 1 2 1 3 6 7 2 0 4 7 3 1 4 9 4 9 0 7 6 5 7 2 1
##   [433] 9 1 6 5 3 0 0 4 9 1 6 1 1 4 7 3 6 5 2 6 3 6 5 2 2 4 1 7 6 7 2 3 2 7 5 7
##   [469] 0 2 2 5 0 9 2 3 7 9 1 1 1 1 4 8 3 5 3 0 0 2 3 2 5 5 6 7 3 0 8 5 9 1 7 5
##   [505] 8 1 0 7 3 8 8 3 1 4 7 7 0 8 2 3 5 3 1 1 4 7 4 9 5 6 1 2 2 1 5 9 2 0 1 3
##   [541] 6 8 0 8 2 9 9 3 2 4 3 0 9 0 8 5 6 1 0 2 9 2 7 6 7 7 6 6 3 6 2 9 6 2 1 0
##   [577] 5 6 9 5 9 8 4 0 4 7 2 8 4 1 7 0 1 3 2 1 5 3 3 4 1 4 5 1 8 3 9 8 9 2 3 8
##   [613] 5 2 1 0 3 0 8 9 3 1 0 9 1 7 3 3 6 3 2 0 9 5 7 9 0 2 3 1 2 9 0 4 0 0 7 3
##   [649] 3 3 5 6 3 8 4 4 0 3 7 1 5 1 8 3 6 7 3 1 4 7 9 1 5 7 0 7 2 3 5 8 5 2 2 2
##   [685] 4 6 0 5 2 9 2 0 1 9 8 8 9 0 3 6 5 8 2 2 0 5 8 9 7 8 6 2 6 4 2 8 4 8 2 0
##   [721] 2 1 9 5 5 9 7 3 3 7 9 3 1 8 1 0 2 7 5 9 0 2 3 2 7 9 1 7 9 2 8 0 3 5 2 6
##   [757] 3 4 9 4 1 1 5 7 4 6 0 1 7 7 3 9 1 1 0 7 5 0 9 6 8 7 0 8 1 9 3 8 9 1 3 3
##   [793] 4 6 2 2 2 1 9 8 6 2 0 1 0 0 0 0 2 7 9 5 3 0 6 1 2 4 8 4 9 8 0 3 8 6 5 5
##   [829] 7 1 1 6 6 6 5 3 8 6 4 6 1 1 8 1 9 4 1 5 2 8 3 6 1 8 6 1 9 3 6 6 6 8 0 0
##   [865] 7 9 2 0 3 2 0 1 0 3 2 8 0 3 2 0 5 6 9 1 8 1 8 3 5 6 7 9 7 4 8 0 3 9 2 8
##   [901] 4 5 9 7 6 3 4 9 9 0 3 4 8 8 7 3 6 1 1 4 9 6 5 9 6 3 2 3 1 9 5 2 4 7 4 4
##   [937] 1 7 6 3 3 2 7 7 9 3 2 8 0 4 4 8 2 5 9 1 3 8 5 2 4 2 1 5 0 0 1 4 1 0 6 7
##   [973] 0 5 2 2 1 2 8 6 3 5 9 0 9 7 9 3 5 1 4 9 8 8 8 4 1 8 0 8 7 2 3 8 3 6 1 3
##  [1009] 6 7 1 0 8 7 5 6 8 2 3 7 5 3 6 9 9 5 1 6 4 6 4 1 4 2 7 9 7 6 4 3 1 7 7 8
```

```
##  [1045] 9 1 2 2 9 6 2 7 1 6 4 4 3 0 5 6 3 9 6 2 1 5 7 6 6 4 4 0 2 1 6 5 2 2 8 3
##  [1081] 5 0 9 5 5 9 2 2 3 6 2 2 0 7 8 4 8 5 0 1 9 6 0 1 9 6 7 7 1 8 1 4 1 4 6 5
##  [1117] 9 2 2 3 0 6 9 1 0 4 4 8 9 3 9 4 0 7 2 1 5 7 2 4 4 8 8 6 0 9 1 7 9 8 1 2
##  [1153] 7 7 1 5 7 2 7 9 0 2 6 7 1 4 4 1 7 8 2 2 1 3 0 3 4 1 6 0 2 9 0 1 4 4 6 7
##  [1189] 8 6 7 3 5 8 0 6 7 8 3 2 4 8 6 2 2 4 1 2 8 9 6 2 9 0 8 6 8 2 5 7 3 7 4 1
##  [1225] 7 8 1 6 0 5 3 2 3 6 6 8 8 9 1 0 1 6 8 0 7 3 3 9 5 7 8 8 6 3 3 5 1 8 7 8
##  [1261] 9 7 2 0 7 6 5 7 2 9 4 5 0 9 7 3 2 1 4 1 6 2 1 1 9 5 4 8 0 3 8 1 5 0 3 4
##  [1297] 7 9 6 4 7 8 0 2 1 1 1 8 3 1 6 6 0 8 7 4 1 7 9 3 4 2 1 4 2 5 9 1 3 1 8 3
##  [1333] 6 1 9 2 1 8 2 3 7 5 0 6 7 9 9 8 3 7 7 8 9 4 2 9 7 1 2 0 1 3 7 3 7 9 7 3
##  [1369] 5 2 4 1 5 6 7 8 2 0 8 4 5 2 6 5 7 1 2 2 9 1 7 5 4 8 9 1 0 3 6 6 5 7 1 9
##  [1405] 9 2 1 8 3 0 7 1 8 0 2 8 5 6 2 1 0 3 5 8 6 5 6 9 5 5 7 1 0 1 8 0 5 9 5 4
##  [1441] 7 5 0 8 1 3 9 3 7 4 1 4 3 4 8 2 0 3 2 9 5 9 9 3 2 8 5 6 5 1 5 7 5 0 7 3
##  [1477] 5 8 7 4 3 8 2 6 4 8 6 3 5 9 3 1 3 0 6 7 5 1 2 6 8 3 4 8 2 0 8 2 0 7 9 2
##  [1513] 9 8 5 6 4 6 5 1 1 2 4 0 8 4 9 7 2 8 3 3 4 6 1 7 8 6 4 3 3 1 0 1 9 5 6 0
##  [1549] 4 6 1 1 6 0 9 1 0 6 4 2 2 7 9 2 0 8 0 2 0 7 3 6 9 3 7 3 2 4 6 7 7 0 7 2
##  [1585] 5 8 8 3 7 6 4 2 0 8 7 9 3 0 3 9 8 3 9 6 4 7 6 3 5 5 6 2 9 3 6 7 6 1 3 9
##  [1621] 1 5 8 3 0 6 3 9 7 1 1 0 6 7 1 9 1 2 0 1 9 8 7 0 2 2 2 9 8 1 5 0 8 9 1 5
##  [1657] 9 4 3 3 6 3 9 2 6 7 8 8 5 2 6 5 4 7 0 8 7 9 0 3 2 3 8 0 5 3 7 0 8 3 1 1
##  [1693] 0 4 2 3 1 0 4 5 9 0 6 8 9 1 4 0 5 1 2 6 3 5 7 3 7 3 3 1 7 9 4 1 4 1 3 7
##  [1729] 3 7 7 1 0 2 9 9 5 5 0 3 0 6 6 6 9 8 2 0 9 8 0 9 7 8 7 6 3 7 8 8 1 3 0 4
##  [1765] 5 9 1 6 7 1 2 6 3 9 4 7 7 3 9 2 7 9 8 8 6 4 0 9 7 9 0 5 3 2 6 2 6 5 0 4
##  [1801] 9 1 6 5 0 4 6 5 0 9 4 3 5 7 6 0 2 9 3 4 6 4 7 3 3 7 7 6 6 5 3 7 6 1 5 0
##  [1837] 1 6 7 9 8 4 2 8 6 1 0 1 7 7 0 1 8 9 4 8 4 7 7 6 4 9 8 7 3 3 1 5 5 7 0 6
##  [1873] 8 2 8 8 2 5 8 2 9 2 7 4 7 7 4 3 7 6 3 3 7 7 2 3 2 4 3 4 1 3 9 4 1 3 5 5
##  [1909] 0 1 2 3 5 1 2 7 1 5 4 0 4 6 9 5 7 6 2 1 4 0 4 2 3 2 3 0 7 6 4 5 2 7 3 0
##  [1945] 9 0 3 8 5 2 7 7 9 2 7 2 8 9 3 5 2 7 2 9 5 2 5 3 5 5 0 6 4 7 8 3 3 4 9 0
##  [1981] 9 2 1 1 5 7 1 7 2 4 2 9 9 5 8 7 0 8 2 6 3 5 9 8 6 2 7 7 5 3 1 8 1 1 7 5
##  [2017] 4 7 7 8 1 1 7 4 3 5 1 8 3 9 3 1 8 5 0 0 5 3 2 1 7 7 1 8 9 3 8 5 3 5 9 1
##  [2053] 2 0 3 4 0 8 6 3 6 3 6 5 7 8 7 0 3 8 9 5 3 9 4 7 1 7 4 7 3 3 7 8 1 9 0 7
##  [2089] 7 6 6 3 3 7 6 8 5 1 0 4 6 6 3 5 4 8 5 3 7 6 5 2 9 9 6 1 8 0 8 4 8 3 2 6
##  [2125] 2 7 0 7 9 5 1 4 1 9 5 1 8 2 0 2 9 6 0 6 1 9 3 7 3 5 9 9 0 9 1 4 8 0 7 4
##  [2161] 7 7 3 8 6 0 3 9 2 5 2 4 6 9 3 5 8 8 7 5 7 0 5 5 0 2 0 4 2 4 0 0 2 9 5 3
##  [2197] 0 7 3 8 0 4 0 8 8 7 4 6 5 1 9 8 2 7 7 6 3 5 1 1 7 7 4 4 5 5 9 6 8 6 6 2
##  [2233] 3 9 2 7 7 6 0 7 4 1 5 0 0 6 0 7 8 6 1 3 2 3 9 1 9 2 0 9 7 8 4 4 8 9 9 1
##  [2269] 7 2 2 6 4 2 3 5 7 2 4 4 0 6 3 0 3 6 9 4 2 5 1 1 3 9 3 3 2 3 1 0 3 0 5 3
##  [2305] 7 4 0 6 7 2 7 7 9 5 0 7 1 8 8 0 1 5 3 1 9 8 2 2 7 6 9 0 4 6 8 4 1 1 2 6
##  [2341] 6 8 2 5 2 6 4 1 8 5 4 6 1 8 7 0 4 9 7 2 6 5 2 0 5 0 0 3 1 3 5 9 5 9 1 4
##  [2377] 9 0 9 9 9 2 4 8 0 7 9 6 3 9 8 7 8 3 0 4 9 9 6 5 2 2 1 1 1 0 7 5 2 7 1 1
##  [2413] 2 8 0 9 1 9 0 6 7 1 1 8 9 0 4 0 0 9 2 3 0 8 2 6 0 5 7 5 8 4 9 1 9 0 8 8
##  [2449] 1 4 8 6 2 3 9 5 7 4 2 2 1 1 5 8 6 0 2 6 0 8 2 4 2 5 3 3 4 1 3 0 6 3 7 6
##  [2485] 8 8 4 7 7 1 1 1 9 6 1 1 3 7 7 8 5 6 1 2 3 1 4 6 7 1 2 6 7 3 4 4 1 6 2 4
##  [2521] 0 2 6 0 2 0 0 6 3 4 9 3 6 4 2 7 5 0 6 4 9 6 7 9 5 4 9 1 9 7 8 3 0 4 2 7
##  [2557] 5 7 2 2 4 6 0 4 0 9 7 9 4 1 5 3 9 1 6 1 0 6 7 2 2 0 2 2 9 2 5 6 4 7 2 1
##  [2593] 0 4 0 9 6 7 1 2 1 6 2 2 2 7 9 7 9 9 8 5 9 0 1 4 2 5 0 5 7 6 6 2 0 1 2 6
##  [2629] 3 1 8 2 0 6 6 2 8 1 8 7 7 8 4 6 6 6 6 4 4 9 9 7 6 6 3 8 9 7 6 9 6 8 3 2
##  [2665] 9 0 8 0 8 4 1 9 8 1 4 8 0 8 0 9 2 4 8 0 8 7 1 9 0 2 2 7 1 3 7 9 3 9 6 3
##  [2701] 8 8 1 7 3 7 9 4 4 4 6 4 7 8 0 1 9 3 7 4 0 8 1 0 0 4 0 1 7 8 3 2 6 8 2 3
##  [2737] 4 4 3 9 2 5 3 9 1 3 6 9 2 5 8 2 5 7 0 4 3 9 8 0 9 7 3 4 1 7 3 5 0 6 8 0
##  [2773] 7 8 4 7 1 0 9 7 8 7 5 0 1 7 7 1 2 5 4 8 3 8 9 6 5 8 2 0 4 0 6 2 8 3 3 8
##  [2809] 4 0 3 4 2 0 0 6 2 0 6 9 0 2 4 8 8 7 4 4 0 8 5 9 5 5 0 8 6 1 6 8 7 7 6 4
##  [2845] 8 0 0 2 5 1 5 0 7 8 4 1 8 7 6 8 1 6 3 4 3 0 9 2 9 0 6 6 6 2 7 7 4 3 6 4
##  [2881] 5 2 1 4 5 2 6 2 3 5 8 7 7 1 6 9 4 6 9 5 8 5 1 9 2 3 7 6 6 4 4 4 5 5 5 3
##  [2917] 1 5 6 9 1 4 5 2 9 3 5 4 1 1 2 3 7 1 5 7 8 2 8 4 4 2 4 6 5 7 0 7 7 3 3 4
##  [2953] 1 8 0 7 2 1 3 7 2 3 0 1 3 3 0 2 8 7 3 8 9 7 0 3 5 2 1 1 7 0 0 7 7 5 7 9
```

```
##  [2989] 9 0 2 3 1 1 3 2 9 5 4 8 5 7 4 0 0 8 2 1 2 4 0 5 7 5 4 2 9 6 8 4 3 2 8 0
##  [3025] 2 4 6 2 8 6 3 5 7 9 2 5 5 7 9 9 6 1 5 1 1 6 3 9 7 4 2 1 1 9 7 1 8 5 3 0
##  [3061] 3 4 6 2 3 3 7 0 3 2 9 4 1 9 7 2 0 2 6 4 2 0 1 3 1 3 5 6 4 9 0 2 3 8 0 4
##  [3097] 0 7 5 1 4 9 4 6 1 3 6 5 7 9 3 9 9 9 1 9 9 2 1 8 7 6 4 7 9 9 7 1 4 6 6 6
##  [3133] 0 1 1 7 3 8 8 5 7 0 4 1 9 7 5 0 8 2 4 4 6 7 7 5 5 1 9 5 0 1 6 9 4 9 9 5
##  [3169] 2 7 8 8 0 7 3 7 2 1 3 1 3 7 2 3 3 8 8 0 7 6 2 2 9 7 4 9 7 3 1 1 2 3 7 4
##  [3205] 0 3 2 0 9 4 6 2 1 4 7 6 4 8 9 4 7 1 0 1 0 7 5 7 8 7 5 0 8 1 3 5 8 3 5 8
##  [3241] 5 5 4 1 3 9 5 6 2 4 0 2 7 0 7 1 8 7 8 6 1 1 9 8 3 6 9 7 9 2 1 0 9 7 5 8
##  [3277] 8 2 3 1 8 7 0 5 7 2 6 5 4 0 2 6 7 0 8 3 5 7 7 3 6 1 1 8 7 2 5 0 3 7 2 0
##  [3313] 0 3 7 8 9 0 8 5 5 0 9 9 1 7 9 6 6 5 8 2 5 3 6 3 2 6 5 3 3 2 8 7 4 4 1 5
##  [3349] 0 8 3 9 6 8 2 9 6 9 0 5 7 5 8 0 2 2 3 7 4 8 1 2 9 1 1 7 1 4 5 6 8 9 1 2
##  [3385] 6 3 1 0 2 9 2 5 1 0 6 2 4 3 7 7 6 3 2 6 3 6 0 5 3 4 5 8 7 3 8 8 4 6 6 1
##  [3421] 3 7 6 9 5 8 0 2 7 3 3 0 4 2 2 3 7 3 5 3 6 9 9 4 6 9 2 9 9 4 5 6 8 3 7 3
##  [3457] 5 1 2 2 7 5 0 5 9 4 2 3 1 4 5 5 2 7 6 1 3 7 4 3 3 7 8 0 5 3 5 2 0 3 0 8
##  [3493] 4 3 3 9 5 3 5 7 4 5 1 9 4 6 4 2 9 8 3 7 8 1 4 4 8 1 3 1 4 3 0 0 9 2 4 1
##  [3529] 7 8 5 9 5 2 6 9 1 9 0 1 7 9 6 5 7 2 8 9 9 2 6 0 1 5 6 8 6 7 3 8 0 3 8 1
##  [3565] 0 2 2 0 2 6 4 1 3 1 3 2 2 1 0 2 4 8 4 2 7 6 9 0 8 4 3 7 3 2 8 5 4 9 8 8
##  [3601] 3 5 0 7 5 4 9 7 7 1 4 4 4 9 9 7 3 3 3 9 9 9 0 1 0 5 6 3 5 5 5 9 9 0 5 7
##  [3637] 5 4 5 6 7 8 2 9 4 5 6 9 3 0 3 4 0 3 2 3 9 3 3 7 1 2 0 7 5 9 7 8 7 4 1 1
##  [3673] 6 0 0 6 3 6 8 8 2 9 7 9 3 1 8 7 3 8 7 0 1 0 2 3 0 2 0 0 7 0 1 0 4 3 3 4
##  [3709] 1 5 4 0 7 6 4 6 7 9 6 6 1 0 5 3 1 8 5 0 8 8 3 7 3 8 5 7 1 9 9 2 9 6 2 3
##  [3745] 4 4 1 4 1 0 2 7 4 5 8 7 4 8 1 0 2 2 0 3 5 0 0 2 2 5 8 3 5 7 9 7 0 1 4 1
##  [3781] 3 9 1 8 9 3 8 3 4 2 7 8 7 4 4 8 9 1 1 2 6 7 5 7 4 8 1 2 7 9 3 7 6 7 5 7
##  [3817] 7 4 7 6 3 6 4 2 4 3 8 5 8 9 3 2 6 6 5 6 7 9 6 0 2 7 8 9 9 7 1 7 0 0 3 4
##  [3853] 0 1 1 1 6 5 8 6 1 2 7 4 0 0 9 1 5 2 9 2 3 2 4 1 3 3 2 4 6 0 9 1 0 8 6 0
##  [3889] 5 0 1 9 3 1 1 8 4 0 1 3 0 8 6 7 6 6 6 5 9 6 9 1 3 7 0 9 3 7 4 9 8 9 3 6
##  [3925] 8 7 5 7 3 8 2 6 8 9 0 0 6 0 8 0 6 3 0 4 1 8 8 3 3 9 3 2 8 0 4 3 2 7 3 7
##  [3961] 2 7 6 8 6 2 3 8 3 0 3 1 3 1 9 6 7 1 3 3 1 4 2 1 4 6 1 4 9 5 7 6 0 5 9 6
##  [3997] 4 6 3 0 5 2 9 1 1 0 1 1 4 6 5 1 9 4 9 8 1 8 6 4 4 1 2 1 1 0 6 1 6 9 9 6
##  [4033] 1 5 0 1 2 9 3 9 9 4 6 0 7 9 7 3 4 8 1 8 3 4 1 1 0 1 5 0 1 3 6 5 2 8 4 9
##  [4069] 2 0 9 8 2 1 5 0 6 5 1 6 7 0 6 9 1 3 4 5 7 6 7 3 3 7 0 8 1 2 4 5 4 1 1 9
##  [4105] 8 1 9 9 0 8 7 5 2 4 2 9 3 8 0 8 2 1 9 7 4 1 4 8 6 3 9 0 7 1 8 1 4 0 2 7
##  [4141] 3 2 3 7 2 9 1 1 5 7 1 9 5 4 6 9 3 1 6 0 9 1 3 9 2 8 8 4 3 8 2 2 5 3 7 8
##  [4177] 7 6 1 7 2 3 5 2 8 4 9 6 6 5 3 6 8 7 3 5 2 7 9 0 0 9 4 1 9 7 1 9 2 7 8 9
##  [4213] 2 9 1 1 6 3 0 5 5 2 5 3 3 8 2 7 0 2 2 7 3 2 7 3 7 6 7 6 6 3 9 5 2 5 8 8
##  [4249] 5 4 0 1 7 8 0 8 6 6 1 7 3 2 3 4 4 3 4 3 9 2 3 2 1 0 0 9 1 3 0 0 8 6 1 9
##  [4285] 2 4 7 4 6 4 2 1 2 8 5 5 7 2 7 9 6 1 9 5 6 2 5 3 5 7 5 7 9 1 2 9 3 8 0 0
##  [4321] 0 3 8 7 1 7 3 8 3 1 0 5 4 4 7 3 2 6 2 1 7 3 9 1 6 5 1 5 3 6 0 6 6 8 6 4
##  [4357] 1 0 6 9 4 1 7 3 1 9 4 4 3 9 7 6 4 9 2 3 0 4 1 6 5 9 4 7 6 9 7 0 0 8 1 0
##  [4393] 8 1 1 6 8 4 8 5 7 9 6 1 6 9 9 7 2 5 9 8 2 2 8 2 9 3 1 1 8 3 4 8 4 2 7 3
##  [4429] 2 6 7 6 1 5 2 5 8 0 4 8 2 9 5 7 8 2 8 2 2 6 4 2 8 3 6 4 9 5 6 0 2 0 0 9
##  [4465] 8 2 7 3 7 3 3 0 4 2 8 5 0 5 0 5 1 7 2 6 4 1 7 2 7 6 3 9 9 1 0 7 9 4 2 8
##  [4501] 7 2 4 6 0 4 6 7 6 1 4 7 2 6 3 0 6 1 7 2 6 0 5 4 4 0 7 3 9 8 6 3 1 4 9 6
##  [4537] 1 9 3 7 7 3 4 7 7 6 1 3 4 2 1 4 3 0 7 7 8 9 2 3 7 3 2 2 4 1 5 8 7 8 8 6
##  [4573] 6 8 1 0 4 3 7 9 4 6 7 4 5 1 5 7 5 6 5 7 0 2 7 2 6 4 6 3 7 7 9 4 3 5 5 2
##  [4609] 9 8 3 8 2 5 8 1 4 7 4 5 6 6 3 6 2 3 8 2 2 5 0 8 9 9 6 3 1 4 6 3 1 5 1 7
##  [4645] 3 8 3 3 9 2 9 9 2 8 6 6 9 3 9 2 3 5 0 7 9 6 8 8 9 3 7 8 2 5 6 0 7 2 9 0
##  [4681] 4 2 7 5 9 1 4 1 1 2 2 6 5 4 7 0 8 3 5 9 1 7 7 6 1 4 1 0 3 9 7 5 7 7 4 5
##  [4717] 3 6 0 9 8 7 4 0 0 3 1 6 2 0 8 7 0 1 5 5 5 5 3 0 7 3 7 6 4 8 0 8 6 0 1 9
##  [4753] 2 6 8 7 4 4 6 9 7 0 6 3 8 9 0 6 3 9 1 3 2 7 8 9 2 0 3 0 1 4 1 2 1 7 9 1
##  [4789] 9 3 1 1 7 7 2 2 6 3 6 0 0 3 1 1 3 6 3 0 0 5 3 9 5 7 2 5 2 8 8 2 2 8 3 8
##  [4825] 7 6 8 0 6 5 9 6 8 8 1 2 6 9 4 6 4 1 6 5 6 3 5 4 2 4 3 1 5 4 2 4 5 5 9 2
##  [4861] 8 5 2 9 1 7 5 8 5 8 9 3 9 8 5 9 6 8 1 5 2 0 6 7 8 6 7 0 9 5 3 9 1 4 3 3
##  [4897] 5 1 9 7 2 1 0 4 1 1 5 8 5 5 3 3 8 9 6 1 9 6 6 2 4 4 8 6 3 6 1 5 2 0 2 7
```

```
##  [4933] 9 0 9 9 3 6 8 6 3 4 0 2 1 6 4 7 1 5 6 6 2 5 8 9 2 3 0 4 0 1 6 5 4 7 8 9
##  [4969] 4 2 9 1 0 7 5 7 2 1 4 3 1 5 4 0 1 5 4 2 0 4 3 1 8 1 6 1 9 7 2 9 6 4 1 9
##  [5005] 4 9 9 9 9 6 9 7 5 2 3 3 3 3 7 5 8 0 0 3 5 8 1 6 7 0 1 1 4 0 9 3 7 1 9 2
##  [5041] 0 1 7 4 6 9 6 2 9 7 2 7 0 7 5 2 0 7 4 0 6 4 0 6 1 2 0 1 4 4 9 1 0 7 2 7
##  [5077] 7 3 9 9 0 5 4 4 2 8 3 5 9 9 9 7 5 1 5 1 0 8 1 1 0 1 3 8 1 8 2 1 8 3 0 8
##  [5113] 8 2 8 2 0 4 5 2 8 0 1 4 1 8 0 5 2 1 1 9 4 3 8 2 3 3 2 2 9 3 6 7 7 8 3 4
##  [5149] 6 8 5 7 5 1 7 3 3 2 8 2 1 9 4 9 5 4 1 9 0 4 2 1 3 7 3 8 1 1 8 1 8 2 9 4
##  [5185] 4 4 7 4 9 8 7 5 4 0 1 3 3 7 7 0 7 5 1 0 7 9 5 4 9 3 3 1 3 1 7 2 4 7 2 4
##  [5221] 6 0 6 7 2 5 7 6 8 5 1 0 1 7 4 6 1 7 0 9 4 2 7 3 4 7 8 4 9 7 6 3 0 8 7 6
##  [5257] 4 4 7 0 7 9 4 5 3 7 6 7 4 8 9 1 6 1 4 8 6 5 0 4 2 4 0 2 0 7 5 4 2 8 0 2
##  [5293] 2 1 3 8 5 9 7 5 7 9 6 2 9 8 1 4 3 2 6 0 1 2 8 2 2 3 2 4 6 5 3 8 1 2 2 5
##  [5329] 2 0 6 0 7 9 0 1 8 4 5 8 2 3 3 8 0 1 0 6 7 1 6 2 0 5 3 7 8 0 4 8 2 6 6 0
##  [5365] 7 8 2 1 2 2 2 2 6 6 4 1 4 6 1 8 7 2 6 6 5 9 7 8 8 2 3 3 8 4 2 7 6 9 8 4
##  [5401] 0 0 6 6 3 1 0 0 3 6 0 1 6 7 0 7 4 0 4 5 4 8 3 1 5 0 3 8 5 3 3 8 1 7 3 7
##  [5437] 2 8 7 5 7 3 1 7 1 4 5 7 6 0 4 3 7 7 5 5 2 0 7 1 1 3 7 4 0 6 3 9 2 2 7 4
##  [5473] 6 6 6 0 3 0 1 4 4 1 4 2 1 2 7 5 0 2 8 8 9 3 2 1 3 1 5 9 1 9 8 7 3 8 3 2
##  [5509] 8 6 6 8 6 0 9 4 1 3 9 0 3 5 7 1 1 9 8 8 8 4 8 6 7 2 9 8 7 2 2 3 7 3 2 9
##  [5545] 5 2 4 4 2 0 8 9 8 2 4 1 3 6 3 1 4 9 1 8 5 1 5 7 3 4 7 6 9 6 7 6 3 7 8 0
##  [5581] 7 9 4 4 5 3 8 7 1 3 0 9 3 1 7 5 2 3 1 5 7 9 3 8 9 1 9 1 1 8 0 2 6 5 4 3
##  [5617] 4 5 8 5 1 0 0 1 9 0 4 1 7 8 0 6 6 4 0 6 2 7 3 0 4 3 4 3 8 8 7 8 7 4 3 3
##  [5653] 0 7 1 8 1 0 2 2 3 9 7 0 8 9 5 6 6 2 3 4 0 7 1 5 3 3 5 3 0 6 8 7 9 7 5 3
##  [5689] 0 7 8 2 0 8 9 9 5 7 5 7 4 9 5 3 9 3 3 1 8 9 6 7 8 3 4 9 9 0 6 9 3 2 6 8
##  [5725] 6 1 6 9 0 7 1 6 4 1 3 0 7 5 1 4 5 9 7 6 9 2 9 6 2 5 2 9 1 3 6 4 7 9 6 9
##  [5761] 3 3 7 9 9 0 8 7 4 7 2 7 3 3 3 5 2 9 6 7 8 8 8 5 7 2 1 1 9 6 8 1 7 7 3 1
##  [5797] 9 6 9 2 5 0 0 2 7 5 6 5 2 9 5 1 8 8 9 0 5 8 5 5 1 6 2 7 8 3 2 6 6 2 0 2
##  [5833] 9 2 7 9 0 4 6 9 4 7 6 1 2 9 1 3 7 9 8 5 0 2 3 6 8 6 4 6 4 8 5 9 8 3 0 8
##  [5869] 7 6 7 9 3 1 1 0 6 3 2 3 9 2 9 6 3 6 1 0 2 5 7 9 8 1 5 1 0 6 0 6 9 0 4 4
##  [5905] 3 3 6 7 1 2 0 7 9 6 3 7 0 3 2 9 4 2 5 3 4 2 4 9 8 8 5 8 6 5 8 7 9 7 5 6
##  [5941] 2 1 2 9 7 1 2 2 9 3 5 6 4 8 6 4 6 2 4 2 2 5 5 6 4 5 3 1 6 7 8 2 1 4 1 9
##  [5977] 4 6 1 2 3 9 7 2 6 3 6 3 9 1 4 3 4 7 0 7 9 1 0 4 5 9 5 5 9 6 6 4 1 3 9 4
##  [6013] 8 5 5 4 1 0 9 2 2 5 4 8 5 0 0 8 7 5 5 5 6 9 5 6 5 2 8 1 6 0 1 6 3 2 6 7
##  [6049] 7 5 1 1 8 4 0 5 6 9 2 8 8 7 9 2 3 2 0 1 0 5 1 1 0 2 4 2 7 0 8 7 2 8 2 3
##  [6085] 8 9 6 9 5 3 1 2 6 8 1 6 7 3 7 9 6 2 8 4 9 5 3 5 5 2 1 0 6 1 1 5 4 3 9 9
##  [6121] 3 6 1 7 8 5 6 7 7 0 1 7 1 3 0 7 6 5 9 5 4 7 0 0 7 8 6 6 1 3 8 0 9 8 2 6
##  [6157] 7 6 1 4 1 8 4 3 6 8 5 3 9 6 5 0 5 9 4 6 9 3 9 3 0 3 3 2 2 8 1 5 8 9 1 2
##  [6193] 0 6 0 6 5 0 6 5 0 7 1 1 0 7 5 8 1 1 3 5 3 9 0 3 7 4 8 6 9 3 8 8 2 1 9 8
##  [6229] 6 9 0 7 1 4 7 0 2 4 5 4 7 0 3 6 2 7 4 1 3 3 9 2 4 9 9 4 0 1 1 7 3 1 8 9
##  [6265] 4 4 7 2 1 0 1 1 4 5 4 0 5 8 7 4 7 8 1 3 8 0 5 0 8 1 9 8 2 2 3 1 3 9 9 4
##  [6301] 1 9 0 3 9 0 5 3 2 3 3 0 1 1 0 7 1 2 4 0 3 0 5 8 8 7 3 8 1 9 1 2 7 6 0 4
##  [6337] 0 1 8 3 1 4 9 1 0 7 7 0 3 6 5 1 1 7 3 5 9 2 9 9 3 8 0 7 5 0 3 4 4 9 9 7
##  [6373] 1 0 1 1 6 3 0 3 2 6 3 8 9 8 2 6 6 3 6 2 0 9 6 0 6 4 1 1 5 7 1 5 6 3 6 3
##  [6409] 3 4 1 7 8 1 7 8 2 0 2 6 2 4 2 0 7 4 7 8 7 3 1 6 0 6 7 4 9 7 7 9 0 0 2 4
##  [6445] 7 9 2 6 2 5 1 9 3 3 5 4 4 2 0 1 9 2 2 2 5 5 8 0 8 8 1 1 1 1 0 8 0 4 3 9
##  [6481] 2 5 9 0 7 0 5 3 2 4 1 1 4 5 6 6 7 1 3 4 8 4 6 4 0 0 0 8 6 6 6 9 0 6 2 7
##  [6517] 1 7 5 0 6 1 9 5 4 5 3 0 2 6 2 2 8 1 2 7 8 5 7 4 1 6 7 3 0 9 8 5 8 6 1 8
##  [6553] 7 6 3 3 6 1 9 1 3 2 2 1 6 7 5 9 1 5 4 5 0 8 1 7 9 1 2 5 2 8 4 6 7 4 0 4
##  [6589] 1 9 5 1 7 9 8 8 0 6 4 4 3 4 7 4 6 3 0 3 6 1 1 2 9 0 2 4 3 5 0 6 7 3 3 4
##  [6625] 1 4 5 8 1 8 6 0 4 7 2 3 9 5 0 5 2 3 4 9 2 3 7 7 7 3 3 7 6 1 4 2 8 2 5 3
##  [6661] 2 9 2 7 7 1 6 5 9 6 4 6 0 0 4 4 3 2 1 2 3 1 0 5 4 1 2 1 5 2 6 0 8 5 7 2
##  [6697] 4 5 0 3 1 6 5 8 4 6 1 3 9 0 7 1 4 8 8 8 1 4 7 3 8 9 6 0 8 1 5 7 2 1 4 3
##  [6733] 1 9 4 3 8 5 0 2 4 8 7 1 6 4 3 5 4 6 1 5 0 4 7 2 8 1 6 7 1 3 2 9 9 0 4 9
##  [6769] 1 5 7 3 6 1 0 1 0 2 6 8 7 5 1 9 3 6 0 2 7 5 4 0 5 6 4 1 7 2 6 4 0 8 5 6
##  [6805] 7 1 1 8 2 2 3 4 4 4 0 0 5 8 0 6 5 1 1 3 0 8 8 1 4 9 4 6 2 1 1 2 5 4 1 8
##  [6841] 3 4 1 2 5 2 5 6 2 4 6 8 2 7 0 6 8 8 5 8 8 7 5 5 3 3 7 1 4 9 9 9 1 1 7 7
```

```
##  [6877] 3 9 9 5 7 8 0 5 9 7 9 6 5 2 7 0 3 8 4 6 2 4 5 5 5 5 5 8 7 5 7 6 6 0 2 0
##  [6913] 7 1 0 2 1 7 4 8 0 7 6 6 5 3 4 2 0 8 5 3 3 8 7 4 3 6 8 4 1 1 2 0 8 4 1 5
##  [6949] 1 1 0 7 7 1 9 4 3 1 9 9 3 0 2 5 6 2 5 9 4 6 8 2 7 6 7 2 0 1 5 2 1 6 2 2
##  [6985] 8 9 2 6 9 6 8 3 8 8 2 9 4 1 8 9 7 4 0 1 8 3 7 0 5 8 6 6 8 7 6 5 8 8 1 2
##  [7021] 9 0 2 9 0 9 6 5 6 6 1 5 6 5 8 8 4 2 5 1 4 5 8 5 2 3 8 8 4 7 3 5 2 4 9 1
##  [7057] 8 3 7 2 9 2 6 2 5 3 2 0 9 3 9 9 8 2 0 7 9 4 0 7 7 7 8 4 7 2 3 4 2 8 9 9
##  [7093] 8 1 1 4 4 3 9 0 1 5 0 4 7 1 9 2 9 6 3 6 4 8 3 4 2 2 1 8 4 1 5 8 9 7 8 5
##  [7129] 6 4 6 2 0 1 3 3 0 2 3 8 9 6 5 0 2 2 2 1 5 4 0 1 2 1 9 7 8 4 3 8 9 8 3 5
##  [7165] 6 0 1 2 9 9 4 1 5 3 8 9 1 1 3 3 8 8 7 8 1 7 6 1 3 8 8 5 2 4 2 8 2 5 7 6
##  [7201] 2 3 3 2 1 1 3 3 4 4 5 4 2 0 4 6 3 3 1 3 4 8 8 2 7 1 4 6 9 9 9 5 9 8 5 4
##  [7237] 4 7 8 2 4 1 2 2 7 8 7 5 3 5 4 7 5 7 9 7 8 0 4 6 7 1 9 8 4 0 9 8 0 4 4 6
##  [7273] 1 1 6 0 0 6 4 9 0 7 8 9 7 6 6 3 7 5 2 3 0 4 1 5 3 2 2 5 5 1 9 7 4 2 8 5
##  [7309] 0 6 8 6 6 5 7 6 8 4 0 1 7 2 0 8 7 4 8 3 1 8 4 3 6 0 1 3 9 3 9 0 7 2 8 8
##  [7345] 9 3 1 9 8 3 8 4 6 0 4 2 4 8 7 0 0 0 3 7 8 5 5 5 4 2 1 9 7 0 9 8 1 1 5 6
##  [7381] 4 5 5 3 1 3 0 8 5 9 7 1 0 1 9 8 4 3 6 1 1 8 3 9 7 0 0 9 2 7 8 8 6 0 8 5
##  [7417] 8 8 4 8 2 4 8 8 9 5 9 7 0 0 3 5 6 1 8 3 7 0 1 7 0 8 2 0 5 8 1 0 3 0 0 2
##  [7453] 2 8 5 4 1 4 6 4 6 9 1 4 2 2 3 1 6 6 8 1 0 1 8 8 7 9 3 4 6 2 9 0 3 0 2 3
##  [7489] 9 4 7 7 7 9 6 9 4 5 7 5 8 3 8 0 6 9 5 5 2 9 1 7 1 4 0 5 9 0 9 9 5 9 4 4
##  [7525] 1 2 7 8 3 2 2 1 0 7 4 9 1 4 8 7 9 5 5 4 2 8 0 1 9 1 4 8 7 3 8 1 7 8 6 2
##  [7561] 0 8 4 0 4 6 0 0 2 1 1 2 0 6 5 9 7 0 0 0 7 9 0 4 7 9 5 0 1 7 2 7 4 1 2 4
##  [7597] 7 2 5 8 7 8 2 9 6 7 3 6 2 5 8 5 5 8 0 0 2 8 8 6 3 4 9 1 0 4 4 7 5 4 4 8
##  [7633] 9 1 8 3 9 4 4 4 7 3 4 1 7 4 3 2 1 0 4 6 4 6 4 0 1 8 2 9 4 1 0 4 0 1 3 0
##  [7669] 1 2 5 6 2 2 7 7 4 4 0 3 4 1 1 2 6 9 9 1 7 1 8 7 5 6 2 6 4 4 0 2 0 5 4 8
##  [7705] 2 0 1 4 5 0 1 4 2 7 0 0 0 5 3 5 1 9 1 4 0 1 9 7 5 5 0 1 0 7 3 9 2 9 6 3
##  [7741] 1 9 0 9 1 8 3 7 5 6 0 2 6 9 1 4 4 0 0 0 3 7 0 0 3 5 6 7 2 0 7 6 6 5 1 0
##  [7777] 3 1 2 7 3 0 2 9 0 9 0 8 4 1 8 5 2 1 1 0 5 7 1 4 0 4 7 6 3 3 1 4 3 4 4 7
##  [7813] 9 4 1 1 4 2 0 4 1 1 5 4 1 5 8 2 2 6 8 1 3 5 3 7 1 7 2 3 3 5 8 1 2 3 6 0
##  [7849] 4 6 3 4 2 1 8 4 4 4 2 3 5 1 1 1 4 5 1 4 1 5 3 5 1 0 0 3 5 0 2 5 0 2 6 0
##  [7885] 5 6 1 8 6 4 9 2 4 4 7 5 6 7 8 4 8 3 4 6 9 1 8 7 0 0 8 0 4 8 1 8 0 7 3 3
##  [7921] 8 6 2 1 1 1 6 5 9 1 0 4 5 1 7 2 4 8 1 9 5 1 0 5 0 7 6 7 1 1 5 6 7 3 8 7
##  [7957] 6 6 2 8 1 7 7 9 8 2 6 2 9 3 0 0 6 2 1 9 2 1 3 7 2 1 7 1 0 1 3 3 1 4 3 1
##  [7993] 1 8 6 0 0 5 0 2 9 7 3 5 2 3 9 9 6 4 8 4 6 9 7 1 7 2 3 3 4 1 1 2 1 9 8 2
##  [8029] 0 4 0 4 6 6 0 8 6 6 8 6 0 6 5 7 8 7 8 3 5 2 8 5 0 2 4 6 8 4 4 8 4 9 3 8
##  [8065] 0 6 0 1 6 3 1 0 9 8 5 0 1 7 3 6 0 9 4 5 7 5 3 7 0 3 3 1 8 1 0 0 7 0 6 6
##  [8101] 0 1 0 6 7 3 3 0 9 1 9 0 0 7 9 2 4 6 6 9 5 1 2 0 4 9 3 3 5 1 0 9 7 6 2 3
##  [8137] 7 1 3 4 1 6 6 3 4 2 9 5 9 3 9 4 7 7 2 1 7 0 0 5 1 6 4 2 4 3 6 6 8 1 5 7
##  [8173] 9 5 2 1 7 2 7 6 1 6 1 6 1 0 9 7 1 6 4 4 8 1 9 1 5 4 6 0 3 5 2 8 6 9 8 8
##  [8209] 2 1 8 9 9 0 4 1 7 7 3 2 7 5 3 8 3 5 5 1 2 2 1 6 5 8 2 3 6 9 1 4 5 6 4 6
##  [8245] 8 7 5 6 8 6 5 5 6 5 4 5 4 5 1 0 3 5 8 2 3 0 1 9 4 3 2 8 8 6 8 3 7 5 5 5
##  [8281] 7 8 9 4 7 4 4 7 8 2 2 6 3 8 7 9 8 2 2 5 6 1 2 7 0 5 0 5 7 2 0 8 3 9 6 0
##  [8317] 3 9 2 7 6 4 1 9 6 4 4 7 6 9 3 4 6 2 7 9 3 4 1 9 7 3 7 1 1 8 1 1 7 4 1 0
##  [8353] 6 4 1 2 2 6 7 3 9 6 1 7 5 8 7 4 3 9 6 0 9 3 3 1 0 0 6 5 0 4 1 5 7 2 1 2
##  [8389] 0 5 2 4 6 9 6 7 1 3 9 8 9 8 5 2 1 2 3 7 8 2 4 7 0 8 8 4 7 2 9 4 1 0 2 8
##  [8425] 6 8 2 8 0 6 1 1 0 3 2 6 1 2 5 0 6 9 0 7 3 6 9 0 0 8 1 7 6 8 6 9 0 8 0 8
##  [8461] 5 2 1 4 6 0 2 5 6 6 0 4 2 7 6 3 2 2 3 2 7 9 7 2 1 3 4 9 6 2 6 6 7 5 1 1
##  [8497] 1 5 6 9 1 1 6 0 4 9 6 3 3 4 6 0 1 5 3 9 7 2 4 1 2 3 7 1 1 7 6 4 3 7 7 3
##  [8533] 6 2 6 7 0 1 5 6 0 3 6 2 4 3 1 5 5 5 7 8 0 2 0 6 0 1 0 9 1 8 4 7 9 9 4 3
##  [8569] 6 2 1 2 6 4 4 4 6 7 9 5 1 8 7 6 0 0 1 8 7 1 8 7 1 7 4 0 4 6 8 4 6 8 7 6
##  [8605] 2 4 4 4 0 6 7 6 5 5 6 8 0 1 2 2 1 7 3 4 8 9 0 0 8 4 1 6 8 3 9 9 7 6 1 4
##  [8641] 6 1 7 0 8 4 2 7 5 7 8 1 3 6 0 4 8 8 4 1 0 4 9 6 2 6 0 4 8 6 5 9 7 4 2 2
##  [8677] 1 0 1 9 1 7 1 5 1 7 7 8 9 2 3 2 4 9 3 1 7 7 4 6 7 1 7 0 7 4 5 8 9 9 3 5
##  [8713] 4 4 3 3 1 5 8 4 8 5 8 3 4 3 2 6 6 7 9 3 7 1 9 5 7 6 3 1 2 4 7 6 3 9 9 7
##  [8749] 5 1 7 8 6 1 1 2 9 9 1 7 6 2 5 5 2 9 7 6 2 7 9 3 2 1 9 9 7 2 0 1 2 9 8 4
##  [8785] 7 1 6 7 7 3 1 2 6 2 2 1 6 0 0 5 8 3 0 4 8 2 9 4 4 7 6 6 1 8 3 8 2 8 5 7
```

```
##  [8821] 3 8 6 1 2 4 1 1 6 1 7 2 3 9 7 3 2 4 0 9 7 4 4 7 5 2 6 3 8 9 6 1 8 2 8 3
##  [8857] 3 6 3 7 8 9 4 5 3 4 9 3 7 8 4 5 3 4 4 7 6 6 0 5 2 1 7 3 4 2 0 3 0 4 6 6
##  [8893] 8 6 6 8 8 4 5 4 6 1 4 6 3 7 4 8 4 0 4 6 3 9 2 7 5 7 8 1 7 1 9 2 4 2 9 6
##  [8929] 9 2 0 3 9 8 8 7 2 4 4 5 7 7 8 1 7 4 1 0 9 9 6 0 3 0 1 3 3 9 1 3 0 6 6 4
##  [8965] 7 9 0 7 0 4 5 7 5 0 6 1 6 0 5 3 3 4 1 1 2 1 0 5 8 2 4 0 0 3 0 7 5 6 8 8
##  [9001] 8 3 9 3 8 4 7 3 3 6 5 4 5 3 7 4 1 3 3 2 9 0 4 9 6 4 4 7 9 1 1 7 4 4 1 3
##  [9037] 3 1 9 8 1 1 3 9 0 0 1 8 5 0 0 3 5 8 8 0 3 7 0 6 5 0 0 6 4 3 7 6 9 7 7 9
##  [9073] 1 7 6 8 3 2 1 6 5 1 7 7 8 1 8 6 9 8 4 1 1 7 2 6 1 6 6 1 3 3 7 3 6 5 5 2
##  [9109] 0 7 5 2 0 9 8 7 8 2 5 4 1 5 4 5 7 0 1 3 4 6 4 0 6 5 8 8 9 1 0 8 6 2 0 9
##  [9145] 4 2 0 7 9 0 2 5 9 2 5 6 9 0 8 8 2 4 0 7 4 8 2 5 9 7 4 1 2 3 5 3 7 7 2 6
##  [9181] 1 9 2 9 7 7 4 4 2 6 3 4 9 1 4 8 7 2 1 8 0 4 6 8 0 8 4 2 3 4 2 3 4 9 4 3
##  [9217] 0 1 1 6 3 6 2 7 0 0 3 0 6 2 4 2 4 2 4 9 7 1 2 0 7 7 1 8 9 9 1 1 1 6 1 1
##  [9253] 6 8 9 4 3 8 3 8 3 7 4 5 9 0 0 3 5 6 1 6 5 8 0 0 2 2 3 0 9 3 5 3 5 7 5 1
##  [9289] 9 8 8 7 1 1 1 1 8 6 7 4 0 5 6 4 0 7 7 7 1 1 7 9 1 6 8 7 0 8 9 8 1 5 2 6
##  [9325] 7 1 7 3 8 4 0 5 2 6 1 8 4 4 9 1 7 5 4 1 4 3 8 8 3 8 1 3 1 8 6 8 5 2 1 6
##  [9361] 8 1 3 1 2 9 1 5 3 7 6 2 5 6 4 3 6 4 0 2 8 4 0 7 3 9 9 5 1 4 3 6 6 0 7 7
##  [9397] 3 9 4 4 3 4 4 8 6 7 9 4 3 7 6 5 6 6 2 3 1 2 8 5 0 3 3 7 3 6 2 4 0 5 2 4
##  [9433] 0 4 1 3 0 2 0 3 2 9 4 8 8 3 3 6 6 3 4 3 1 4 4 0 0 6 4 1 3 3 8 6 9 8 9 3
##  [9469] 1 2 3 1 5 8 2 9 9 3 5 1 3 4 4 2 3 9 2 4 3 2 0 4 7 1 0 0 4 5 2 4 5 3 1 0
##  [9505] 8 3 8 8 0 4 2 0 9 4 6 9 7 9 7 8 3 9 3 8 8 7 0 0 9 6 3 4 7 2 8 1 3 9 1 0
##  [9541] 4 8 4 4 6 3 7 3 4 8 9 5 5 9 2 7 3 8 6 1 5 0 3 2 5 5 4 6 3 8 7 8 8 9 5 6
##  [9577] 6 4 6 9 6 2 8 8 9 2 5 9 0 0 1 4 7 2 2 2 6 8 2 0 0 5 3 1 5 4 2 9 6 8 4 3
##  [9613] 4 2 2 9 4 7 9 6 5 8 4 5 2 2 9 3 9 9 5 9 4 8 0 8 2 1 9 8 9 6 6 2 6 1 1 8
##  [9649] 8 0 8 7 8 0 5 6 0 4 4 7 2 9 7 1 7 7 8 0 3 7 4 0 7 4 4 8 6 1 8 1 1 3 0 2
##  [9685] 8 6 6 0 4 3 8 2 0 8 2 7 4 9 7 3 2 2 1 9 3 1 2 3 6 7 8 0 3 2 0 7 1 0 8 8
##  [9721] 6 4 3 4 8 0 9 1 8 6 4 7 4 5 3 9 8 7 7 8 6 4 1 8 0 6 7 9 3 6 8 6 7 3 7 6
##  [9757] 8 1 9 2 2 6 2 5 0 1 8 1 3 7 3 8 5 1 7 3 7 1 5 7 3 9 9 3 1 1 0 0 5 6 8 9
##  [9793] 9 4 8 8 7 0 5 5 2 5 1 4 1 4 8 3 6 1 1 6 4 2 5 5 6 2 5 6 3 3 4 7 2 0 3 1
##  [9829] 1 2 0 2 1 7 9 7 6 6 0 6 8 7 4 1 1 8 9 2 2 0 9 3 7 6 0 5 0 4 8 5 0 9 1 1
##  [9865] 2 9 5 1 4 7 7 6 9 1 8 6 8 9 4 4 5 4 1 2 0 7 1 0 0 5 0 9 3 1 3 3 5 0 7 6
##  [9901] 0 4 7 5 6 7 0 5 6 6 9 5 1 8 2 2 4 1 6 8 4 7 0 9 5 8 3 2 0 1 4 0 9 3 8 3
##  [9937] 9 2 4 6 5 7 6 6 6 9 3 2 1 8 9 1 5 3 6 6 7 3 1 3 4 3 2 9 8 2 4 3 9 2 2 3
##  [9973] 4 7 8 7 1 4 2 4 1 3 8 6 9 6 3 5 6 1 5 0 5 7 7 4 2 1 2 8
```

```r
for_submission <- as.matrix(predictions)
```

In this chunk, the test data is loaded into the model. The predictions are put inot an object called predictions that will be used to convert the data into a csv to be uploaded. The k_argmax function is used because of a recent update to keras and this was the function to also call to make it into an object. The data is later transformed into a matrix for ease of conversion. This code was from Maria Grycuk from her guide:https://appsilon.com/r-keras-mnist/.

```r
df <- data.frame("Id" = 1:10000, "Expected" = for_submission)
write.csv(df, "kerassub16.csv", row.names = FALSE)
```

Making it a CSV for submission into kaggle.

Here is our attempts from hard coded in with using the 'getting started code

```r
# library(readr)
# x <- as.matrix(read.csv("mnist_train.csv"))
# y <- as.vector(read.csv("mnist_train_targets.csv")$x)
# x_test <- as.matrix(read.csv("mnist_test.csv"))
```

```
# x <- apply(x, 2, function(y) y / 255)
# x_test <- apply(x_test, 2, function(y) y / 255)
```

Transforming the data to be percentages of brightness rather than 0-255.

```
# library(ggplot2)
# plot_number <- function(fill){
#   df <- data.frame("fill" = fill,
#                    "y" = do.call(c, lapply(28:1, function(i) rep(i, 28))),
#                    "x" = rep(1:28, 28))
#   ggplot(df, aes(x, y, fill = fill)) + geom_tile() + scale_fill_gradient(low = "#FFFFFF", high = "#00
# }
#
#  plot first nine numbers
# cowplot::plot_grid(plotlist = lapply(1:9, function(i) plot_number(x[,i])), nrow = 3, ncol = 3)
```

This visualization chunk could not be used as there were too many errors that could not be solved.

```
# softmax <- function(x) {
#   exp_x <- exp(x - max(x))  # for numerical stability
#   return(exp_x / rowSums(exp_x))
# }
```

Creating the softmax function with assistance from Jacob Bedi.

```
# softmax_prime_slow <- function(x) {
#   p <- softmax(x)
#   return(p * (1 - p))# OR: diag(gx) - gx %o% gx
# }
```

Creating the softmax prime function with assistance from Jacob Bedi.

```
# sigmoid <- function(x){ 1 / (1 + exp(-x))}
# sigmoid_prime <- function(x) x * (1 - x)
# relu <- function(x){
#   x[x < 0] <- 0
#   x
# }
# relu_prime <- function(x){x > 0}
#
# multilayer_perceptron <- function(x, y, epochs, lr, h1, h2, h3){
#   # weight and bias initialization (Glorot initialization)
# w1 <- matrix(runif(nrow(x) * h1, min = -sqrt(6 / (nrow(x) + h1)), max = sqrt(6 / (nrow(x) + h1))), nr
# b1 <- rnorm(h1, sd = 0.1)
# w2 <- matrix(runif(h1 * h2, min = -sqrt(6 / (h1 + h2)), max = sqrt(6 / (h1 + h2))), h1, h2)
# b2 <- rnorm(h2, sd = 0.1)
# w3 <- matrix(runif(h2 * 1, min = -sqrt(6 / (h2 + h3)), max = sqrt(6 / (h2 + h3))), h2, h3)
# b3 <- rnorm(1, sd = 0.1)
#  w4 <- matrix(runif(h3 * 1, min = -sqrt(6 / (h3 + 1)), max = sqrt(6 / (h3 + 1))), h3, 1)
#   b4 <- rnorm(1, sd = 0.1)
#
#
```

```
#   for(epoch in 1:epochs){
#     total_error <- 0
#     for(i in 1:ncol(x)){
#       # forward propagation
#       z1 <- x[,i] %*% w1 + b1
#       a1 <- relu(z1)
#       z2 <- a1 %*% w2 + b2
#       a2 <- sigmoid(z2)
#       z3 <- a2 %*% w3 + b3
#       a3 <- relu(z3)
#       z4 <- a3 %*% w4 + b4
#       a4 <- softmax(z4)
#
#
#       # backpropagation
#       error <- y[[i]] - a4
#       total_error <- total_error + abs(round(error))
#       a4_delta <- error * softmax_prime_slow(a4)
#       a3_delta <- (a4_delta %*% t(w4)) * relu_prime(a3)
#       a2_delta <- (a3_delta %*% t(w3)) * sigmoid_prime(a2)
#       a1_delta <- (a2_delta %*% t(w2)) * relu_prime(a1)
#
#       # update weights and biases
#       w4 <- w4 + lr * t(a3) %*% a4_delta
#       br <- b4 + lr * a4_delta
#       w3 <- w3 + lr * t(a2) %*% a3_delta
#       b3 <- b3 + lr * a3_delta
#       w2 <- w2 + lr * t(a1) %*% a2_delta
#       b2 <- b2 + lr * a2_delta
#       w1 <- w1 + lr * x[,i] %*% a1_delta
#       b1 <- b1 + lr * a1_delta
#     }
#     accuracy <- round((1 - total_error / ncol(x)) * 100, 2)
#     cat(paste0("epoch: ", epoch, ", classification accuracy: ", accuracy, "%\n"))
#   }
#   list(w1 = w1, b1 = b1, w2 = w2, b2 = b2, w3 = w3, b3 = b3, w4 = w4, b4 = b4)
# }
```

The main issue with this chunk is the strain it puts on machines. Specifically on laptops. Run time is anywhere from an hour or 2. This chunk is transformed for three hidden layers with relu and sigmoid. The choice of relu was for how well it did in trials and for its computational simplicity. It also represents sparsity well as this model will deal with many sparse matricies. Sigmoid is built for percentages which is used in this model. The introduction of it in the middle of the function is found to bring the best results for this model. Choice of number of hidden layers and neurons is arbitary and also on trial and error. Many guides recommended large first layers and decrease sizes later on. There were issues with matrix multiplication as non-conformable arguments was common. Jacob Beldi assisted in troubleshooting code. Eventually, there was success in running this chunk, but it consumed a large amount of time to run and was not optimized for any machine.

```
# set.seed(123)
# models <- lapply(0:9, function(digit){
#   multilayer_perceptron(x, as.numeric(y == digit), h1 = 256, h2 = 128, h3 = 64,  lr = 0.001, epochs =
# })
```

This chunk provided the most computational strain, this could not be compiled in a reasonable time. It would take over 3 hours to run. I did run it with 16, 8 and 4. Ideally, more epochs would be make for better optimization but could not be implemented because of time constraints and machine limitations.

```
# Make predictions using softmax models
# predict_softmax_mlp <- function(models, x_test) {
#   pred <- matrix(0, length(models), ncol(x_test))
#   for (i in 1:ncol(x_test)) {
#     for (j in 1:10) {
#       z1 <- x_test[, i] %*% models[[j]]$w1 + models[[j]]$b1
#       a1 <- relu(z1)
#       z2 <- a1 %*% models[[j]]$w2 + models[[j]]$b2
#       a2 <- sigmoid(z2)
#       z3 <- a2 %*% models[[j]]$w3 + models[[j]]$b3
#       a3 <- relu(z3)
#       pred[j, i] <- softmax(z4)
#     }
#   }
#   pred
# }
# pred_test <- apply(predict_mlp(models, x_test), 2, function(y) which.max(y) - 1)
# pred_train <- apply(predict_mlp(models, x), 2, function(y) which.max(y) - 1)
```

The issue in this chunk was it was not able to find object 'a3'. Troubleshooting was not successful in this. It is currently set up to run for 3 hidden layers with two relu activation functions and one sigmoid. This is the point where keras seemed to be the better route to complete this project and build a successful neural net.

```
# df <- data.frame("Id" = 1:10000, "Expected" = pred_test)
# write.csv(df, "example_submission.csv", row.names = FALSE)
```

Not much needs to be said about this chunk as it was used in the actual code above. This is transforming our data from hardcoded model into a CSV.

The first attempt was with hard coded model using the 'getting started' code provided by in the kaggle. This proved to be difficult to implement a 3 hidden layer model to adjust the existing code and add in the new lines for 3 hidden layers. Issues with non-conformable arguments were common, this was fixed with the assistance of Jacob Bedi. The next issue present was runtime and computational strain. Trials could not be done with hard code because of the run time and computational strain. Trials was done using keras and found that 256 relu, 128 sigmoid, and 64 relu and a dropout of .4 provided the best results for the model. Keras proved to be more accurate and much easier to do trials with as the optimizer, forward and back propagation and creating the neurons was giving arguments into a function rather than hard coded and presenting the possibility of issues in errors and syntax issues. The hard code is presented to show effort and understanding of the basics for machine learning. The best accuracy produced was 99.633%.