STAT 6340 Statistical Machine Learning

Bonus Project

Author:
Suarez Rodes, Randy

Supervisor:
Choudhary, Pankaj Ph.D.

May 12, 2021

1 Answers

Question 1.a Fitting a neural network with 1 hidden layer with 512 hidden units and 5 epochs we obtained a training error of 0.9% of and a test error of 1.99%.

Question 1.b Fitting a neural network with 1 hidden layer with 512 hidden units and 10 epochs we obtained a training error of 0.17% and a test error of 1.83%.

Question 1.c Fitting a neural network with 1 hidden layer with 256 hidden units and 5 epochs we obtained a training error of 1.18% and a test error of 2.19%.

Question 1.d Fitting a neural network with 1 hidden layer with 256 hidden units and 10 epochs we obtained a training error 0.32% of and a test error of 1.89%.

Question 1.e Fitting a neural network with 2 hidden layers, each with 512 hidden units and 5 epochs we obtained a training error of 0.60% and a test error of 2.06%.

Question 1.f Fitting a neural network with 2 hidden layers, each with 512 hidden units and 10 epochs we obtained a training error 0.20% of and a test error of 1.68%.

Question 1.g Fitting a neural network with 2 hidden layers, each with 256 hidden units and 5 epochs we obtained a training error of 0.80% and a test error of 2.12%.

Question 1.h Fitting a neural network with 2 hidden layers, each with 256 hidden units and 10 epochs we obtained a training error of 0.26% and a test error of 2.05%.

Question 1.i Adding L2 weight regularization with $\lambda = 0.001$ to the model of 1.a we obtained a training error of 2.68% and a test error of 3.21%.

Question 1.j Adding dropout to the model of 1.a we obtained a training error of 0.26% and a test error of 2.05%.

Question 1.k We present on Table 1 a summary of the results from the previous models. Observe that the models with higher complexity (more hidden units, hidden layers and epochs) exhibit the least training and testing errors. Regularization did not represent a significant improvement, implying that a robust search for the optimal regularization parameter is required. We recommend the model with 1 hidden layer of 256 units trained over 10 epochs. This model obtained the second best test error, 1.89%, falling very close to the 1.83% of a similar model but with 512 hidden units. Therefore we trade a lot of complexity (hence a decrease in variance) for a small decrease in accuracy.

Train_Error	Test_Error	HiddenLayers	Epochs	Regularization	DropOut
0.0090	0.0199	512	5	NO	NO
0.0017	0.0183	512	10	NO	NO
0.0118	0.0219	256	5	NO	NO
0.0032	0.0189	256	10	NO	NO
0.0060	0.0206	512, 512	5	NO	NO
0.0020	0.0168	512, 512	10	NO	NO
0.0080	0.0212	256, 256	5	NO	NO
0.0026	0.0205	256, 256	10	NO	NO
0.0268	0.0321	512	5	L2	NO
0.0026	0.0205	512	5	NO	50%

Table 1: Model Comparison

Question 2.a Applying 4-fold cross validation, we fitted a neural network with 2 hidden layers, each with 64 hidden units, and 200 epochs. We present on Figure 1 the relationship of the validation MAE vs the number of epochs. We can notice that the validation MAE stops improving substantially around epoch 75, reaching the minimum on epoch 141 (marked on Figure 1), therefore we recommend early stopping and we suggest to use 90 epochs for the

current model and all the models onwards.

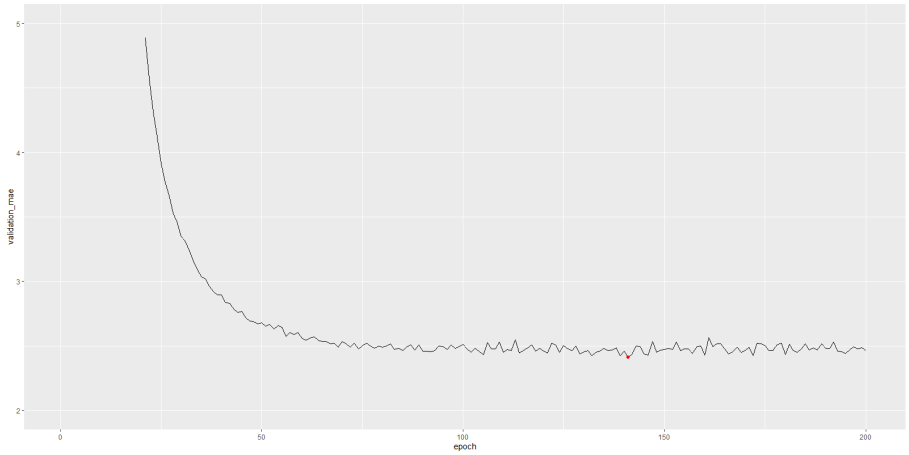

Figure 1: Validation MAE vs Epochs

Fitting a neural network with 2 hidden layers, each with 64 hidden units and 90 epochs we obtained a validation MAE of 2.438 and a test MAE of 2.734.

Question 2.b Fitting a neural network with 1 hidden layer with 128 hidden units and 90 epochs we obtained a validation MAE of 2.344 and a test MAE of 2.801.

Question 2.c Adding L2 weight regularization to the model with 2 hidden layers, each with 64 hidden units and 90 epochs we obtained a validation MAE of 2.482 and a test MAE of 2.662.

Question 2.d Adding L2 weight regularization to the model with 1 hidden layer with 128 hidden units and 90 epochs we obtained a validation MAE of 2.339 and a test MAE of 2.791.

Question 2.e We present on Table 2 a summary of the results from the previous models. Models with higher complexity perform better on the validation data, on the other hand the models with regularization exhibit a clear advantage on the testing data. We recommend the use of the regularized model with two hidden layers, each with 64 hidden units since it offers significant decrease on testing MAE.

Val_MAE	Test_MAE	HiddenLayers	Regularization
2.4382	2.7335	64, 64	NO
2.3436	2.8006	128	NO
2.4816	2.6616	64, 64	L2
2.3394	2.7917	128	L2

Table 2: Model Comparison

2 Code

```
#Seed to replicate results
rdseed=8466
set.seed(rdseed)

library(ggplot2)
library(keras)
library(tensorflow)

#Setting the seed for tensorflow session
tf$random$set_seed(rdseed)

#Experiment 1
print("Experiment 1")

#Number of Epochs for 1a-1d, 1e-1h
epc=c(5,10,5,10)

#Layer sizes to be displayed
layer_sizes=c("512","512","256","256","512, 512","512, 512",
              "256, 256","256, 256","512","512")

#Dataframe to track results
error.df=data.frame(Train_Error=rep(0,10),Test_Error=rep(0,10),
                    HiddenLayers=layer_sizes, Epochs=c(epc,epc,5,5),
                    Regularization=c(rep("NO",8),"L2","NO"),DropOut=c(rep("NO",9),"50%"))

#Importing data set
mnist = dataset_mnist()
train_images = mnist$train$x
train_labels = mnist$train$y
test_images = mnist$test$x
test_labels = mnist$test$y

#Reshaping the data
train_images = array_reshape(train_images, c(60000, 28*28)) # matrix
train_images = train_images/255 # ensures all values are in [0, 1]
test_images = array_reshape(test_images, c(10000, 28*28))
test_images = test_images/255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

####Questions 1a-1d####

#Hidden Layers sizes for 1a-1d
hl_sizes=c(512,512,256,256)

#Fitting models 1a-1d
for(i in seq(1,4))
{
  nn <- keras_model_sequential() %>%
    layer_dense(units = hl_sizes[i], activation = "relu", input_shape = c(28*28) ) %>%
    layer_dense(units = 10, activation = "softmax")

  nn %>% compile(
    optimizer = "rmsprop",
    loss = "categorical_crossentropy",  # loss function to minimize
    metrics = c("accuracy") # monitor classification accuracy
  )
```

```
  nn %>% fit(train_images, train_labels, epochs = epc[i], batch_size = 128)

  metrics_trn=nn %>% evaluate(train_images, train_labels)
  metrics_tst=nn %>% evaluate(test_images, test_labels)

  #Training and Testing Errors
  error.df[i,"Train_Error"]=1-metrics_trn[[2]]
  error.df[i,"Test_Error"]=1-metrics_tst[[2]]
}

####Questions 1e-1h####

#Hidden Layers sizes for 1e-1h
hl1=c(512,512,256,256)
hl2=c(512,512,256,256)

#Fitting models 1e-1h
for(i in seq(1,4))
{
  nn <- keras_model_sequential() %>%
    layer_dense(units = hl1[i], activation = "relu", input_shape = c(28*28)) %>%
    layer_dense(units = hl2[i], activation = "relu") %>%
    layer_dense(units = 10, activation = "softmax")

  nn %>% compile(
    optimizer = "rmsprop",
    loss = "categorical_crossentropy",  # loss function to minimize
    metrics = c("accuracy") # monitor classification accuracy
  )

  nn %>% fit(train_images, train_labels, epochs = epc[i], batch_size = 128)

  metrics_trn=nn %>% evaluate(train_images, train_labels)
  metrics_tst=nn %>% evaluate(test_images, test_labels)

  #Training and Testing Errors
  error.df[i+4,"Train_Error"]=1-metrics_trn[[2]]
  error.df[i+4,"Test_Error"]=1-metrics_tst[[2]]
}


####Questions 1i####

#L2 regularization
nn.reg=keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28*28),
              kernel_regularizer = regularizer_l2(0.001)) %>%
   layer_dense(units = 10, activation = "softmax")

nn.reg %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)

nn.reg %>% fit(train_images, train_labels, epochs = 5, batch_size = 128)

metrics_trn=nn.reg %>% evaluate(train_images, train_labels)
metrics_tst=nn.reg %>% evaluate(test_images, test_labels)

#Training and Testing Errors
error.df[9,"Train_Error"]=1-metrics_trn[[2]]
error.df[9,"Test_Error"]=1-metrics_tst[[2]]
```

```
####Questions 1j####

#Dropout model
nn.drop <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28*28)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 10, activation = "softmax")

nn.drop %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",  # loss function to minimize
  metrics = c("accuracy") # monitor classification accuracy
)

nn.drop %>% fit(train_images, train_labels, epochs = 5, batch_size = 128)

metrics_trn=nn %>% evaluate(train_images, train_labels)
metrics_tst=nn %>% evaluate(test_images, test_labels)

#Training and Testing Errors
error.df[10,"Train_Error"]=1-metrics_trn[[2]]
error.df[10,"Test_Error"]=1-metrics_tst[[2]]


####Questions 1k####
print(error.df)


#Experiment 2
print("Experiment 2")

#Importing data
boston <- dataset_boston_housing()
c(c(train_data, train_targets), c(test_data, test_targets)) %<-% boston

#Layer sizes to be displayed
layer_sizes2=c("64, 64","128","64, 64","128")

#Dataframe to track results
error.df2=data.frame(Val_MAE=rep(0,4),Test_MAE=rep(0,4),
                     HiddenLayers=matrix(layer_sizes2),
                     Regularization=c("NO","NO","L2","L2"))

### Preprocess the data

### Standardize the training and test features
mean <- apply(train_data, 2, mean)
std <- apply(train_data, 2, sd)
train_data <- scale(train_data, center = mean, scale = std)
test_data <- scale(test_data, center = mean, scale = std)

####Questions 2a####

#Helper function for 2a
build_model <- function(){
  # specify the model
  model <- keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu",
                input_shape = dim(train_data[[2]])) %>%
    layer_dense(units = 64, activation = "relu") %>%
    layer_dense(units = 1)
```

```
  # compile the model
  model %>% compile(
    optimizer = "rmsprop",
    loss = "mse",
    metrics = c("mae") # mean absolute error
  )
}


# K-fold CV

k=4
set.seed(rdseed)
indices=sample(1:nrow(train_data))
folds=cut(indices, breaks = k, labels = FALSE)

#Initial number of epochs
num_epochs=200

#MAE history tracker
all_mae_histories=NULL

for (i in 1:k){
  cat("Processing fold #", i, "\n")

  # Prepares the validation data, data from partition #k
  val_indices <- which(folds == i, arr.ind = TRUE)
  val_data <- train_data[val_indices,]
  val_targets <- train_targets[val_indices]

  partial_train_data <- train_data[-val_indices,]
  partial_train_targets <- train_targets[-val_indices]

  #Fitting the model and tracking results
  model <- build_model()

  history <- model %>% fit(
    partial_train_data, partial_train_targets,
    validation_data = list(val_data, val_targets),
    epochs = num_epochs, batch_size = 64, verbose = 0
  )

  mae_history <- history$metrics$val_mae
  all_mae_histories <- rbind(all_mae_histories, mae_history)
}

### Get the mean K-fold validation MAE for each epoch
average_mae_history <- data.frame(
  epoch = seq(1:ncol(all_mae_histories)),
  validation_mae = apply(all_mae_histories, 2, mean)
)

#Best number of epoch for reference purpose
best_epoch=which.min(average_mae_history$validation_mae)

### Plot the history
print(ggplot(average_mae_history,aes(x=epoch,y=validation_mae))+geom_line()+
  ylim(2,5)+geom_point(aes(x=best_epoch,y=validation_mae[best_epoch]),
                       color="red",size=3,shape=20))

#We suggest 90 epochs
final_e=90
```

```
maer=rep(0,k)

for (i in 1:k){
  cat("Processing fold #", i, "\n")

  # Prepares the validation data, data from partition #k
  val_indices <- which(folds == i, arr.ind = TRUE)
  val_data <- train_data[val_indices,]
  val_targets <- train_targets[val_indices]

  partial_train_data <- train_data[-val_indices,]
  partial_train_targets <- train_targets[-val_indices]

  #Fitting the model
  model <- build_model()

  res_i=model %>% fit(
    partial_train_data, partial_train_targets,
    validation_data = list(val_data, val_targets),
    epochs = final_e, batch_size = 16, verbose = 0
  )


  metrics_trn=model %>% evaluate(val_data, val_targets)

  #Results of current fold
  maer[i]=metrics_trn[[2]]

}

#Estimated validation MAE
error.df2[1,"Val_MAE"]=mean(maer)

#Fitting with all training data
model <- build_model()
model %>% fit(train_data, train_targets, epochs = final_e,
              batch_size = 16, verbose = 0)

#Test MAE
metrics_tst=model %>% evaluate(test_data, test_targets)
error.df2[1,"Test_MAE"]=metrics_tst[[2]]



####Questions 2b####

#Helper function for 2a
build_model2 <- function(){
  # specify the model
  model <- keras_model_sequential() %>%
    layer_dense(units = 128, activation = "relu",
                input_shape = dim(train_data[[2]])) %>%
    layer_dense(units = 1)
  # compile the model
  model %>% compile(
    optimizer = "rmsprop",
    loss = "mse",
    metrics = c("mae") # mean absolute error
  )
}


maer=rep(0,k)
```

```
for (i in 1:k){
  cat("Processing fold #", i, "\n")

  # Prepares the validation data, data from partition #k
  val_indices <- which(folds == i, arr.ind = TRUE)
  val_data <- train_data[val_indices,]
  val_targets <- train_targets[val_indices]

  partial_train_data <- train_data[-val_indices,]
  partial_train_targets <- train_targets[-val_indices]

  #Fitting the model
  model <- build_model2()

  res_i=model %>% fit(
    partial_train_data, partial_train_targets,
    validation_data = list(val_data, val_targets),
    epochs = final_e, batch_size = 16, verbose = 0
  )


  metrics_trn=model %>% evaluate(val_data, val_targets)

  #Results of current fold
  maer[i]=metrics_trn[[2]]

}

#Estimated validation MAE
error.df2[2,"Val_MAE"]=mean(maer)

#Fitting with all training data
model <- build_model2()
model %>% fit(train_data, train_targets, epochs = final_e,
              batch_size = 16, verbose = 0)

#Test MAE
metrics_tst=model %>% evaluate(test_data, test_targets)
error.df2[2,"Test_MAE"]=metrics_tst[[2]]


####Questions 2c####

#Helper function for 2c
build_model3 <- function(){
  # specify the model
  model <- keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu",
                input_shape = dim(train_data[[2]]),
                kernel_regularizer = regularizer_l2(0.001)) %>%
    layer_dense(units = 64, activation = "relu",
                kernel_regularizer = regularizer_l2(0.001)) %>%
    layer_dense(units = 1)
  # compile the model
  model %>% compile(
    optimizer = "rmsprop",
    loss = "mse",
    metrics = c("mae") # mean absolute error
  )
}
```

```
maer=rep(0,k)

for (i in 1:k){
  cat("Processing fold #", i, "\n")

  # Prepares the validation data, data from partition #k
  val_indices <- which(folds == i, arr.ind = TRUE)
  val_data <- train_data[val_indices,]
  val_targets <- train_targets[val_indices]

  partial_train_data <- train_data[-val_indices,]
  partial_train_targets <- train_targets[-val_indices]

  #Fitting the model
  model <- build_model3()

  res_i=model %>% fit(
    partial_train_data, partial_train_targets,
    validation_data = list(val_data, val_targets),
    epochs = final_e, batch_size = 16, verbose = 0
  )


  metrics_trn=model %>% evaluate(val_data, val_targets)

  #Results of current fold
  maer[i]=metrics_trn[[2]]

}

#Estimated validation MAE
error.df2[3,"Val_MAE"]=mean(maer)

#Fitting with all training data
model <- build_model3()
model %>% fit(train_data, train_targets, epochs = final_e,
              batch_size = 16, verbose = 0)

#Test MAE
metrics_tst=model %>% evaluate(test_data, test_targets)
error.df2[3,"Test_MAE"]=metrics_tst[[2]]


####Questions 2d####

#Helper function for 2d
build_model4 <- function(){
  # specify the model
  model <- keras_model_sequential() %>%
    layer_dense(units = 128, activation = "relu",
                input_shape = dim(train_data[[2]]),
                kernel_regularizer = regularizer_l2(0.001)) %>%
    layer_dense(units = 1)
  # compile the model
  model %>% compile(
    optimizer = "rmsprop",
    loss = "mse",
    metrics = c("mae") # mean absolute error
  )
}

maer=rep(0,k)
```

```
for (i in 1:k){
  cat("Processing fold #", i, "\n")

  # Prepares the validation data, data from partition #k
  val_indices <- which(folds == i, arr.ind = TRUE)
  val_data <- train_data[val_indices,]
  val_targets <- train_targets[val_indices]

  partial_train_data <- train_data[-val_indices,]
  partial_train_targets <- train_targets[-val_indices]

  #Fitting the model
  model <- build_model4()

  res_i=model %>% fit(
    partial_train_data, partial_train_targets,
    validation_data = list(val_data, val_targets),
    epochs = final_e, batch_size = 16, verbose = 0
  )


  metrics_trn=model %>% evaluate(val_data, val_targets)

  #Results of current fold
  maer[i]=metrics_trn[[2]]

}

#Estimated validation MAE
error.df2[4,"Val_MAE"]=mean(maer)

#Fitting with all training data
model <- build_model4()
model %>% fit(train_data, train_targets, epochs = final_e,
              batch_size = 16, verbose = 0)

metrics_tst=model %>% evaluate(test_data, test_targets)

#Test MAE
error.df2[4,"Test_MAE"]=metrics_tst[[2]]



####Questions 2e####
print(error.df2)
```