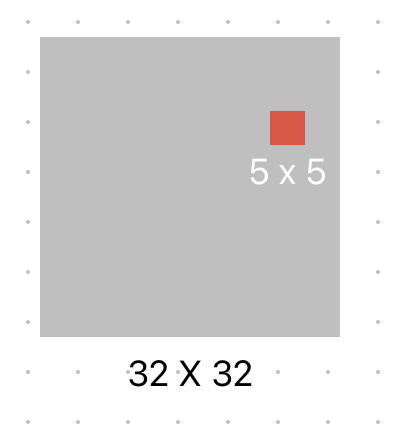
Chapter\_10\_HW

Ryan Gallagher

2023-12-03

## Question 4: Consider a CNN that takes a grayscale images and has a single convolution layer with three convolution filters (without boundary padding)

### (a) Draw a sketch of the input and first hidden layer similar to Figure 10.8



#### (b) How many parameters are in this model?

There are 5x5 weights with only one color channel (grayscale), with three bias terms for the three convolution filters. So, 5x5x1 - 3 = 72 parameters.

#### (c) Explain how this model can be thought of as an ordinary feed-forward neural network with the individual pixels as inputs, andwith constraints on the weights in the hidden units. What are the constraints?

This CNN can be thought of as a specialized FFNN with the constraints of weight sharing and local connectivity. If you were to unroll this CNN into an FFNN, each neuron in the hidden layer would correspond to one application of the filter at a specific location in the input. The neurons corresponding to the same filter would share weights, reflecting the weight-sharing property of the CNN. Each neuron in the hidden layer would be connected to only a subset of the input neurons (those within the area it covers), unlike a typical FFNN neuron, which would connect to all input neurons.

#### (d) If there were no constraints, then how many weights would there be in the ordinary feed-forward neural network in (c)?

Well, there would be an input layer of neurons - one for each pixel. Then, each filter would have since there is a filter cover area. And since there are three of those, we have . Then, since each filter connects to regions of input then each nueron gets unique weights. So, .

## Question 7: Fit a neural network to the Default data. Use a single hidden layer with 10 units, and drop regularization. Have a look at Labs 10.9.1-10.9.2 for guidance. Compare the classification performance of your model with that of linear logistic regression.

library(ISLR2)  
library(torch)  
  
# Load and preprocess the data  
data(Default)  
Default$default = as.numeric(Default$default) - 1 # Convert to numeric  
Default$student = as.numeric(Default$student) - 1 # Convert to numeric  
  
features = Default[, c("balance", "income", "student")]  
labels = Default$default  
  
features = torch\_tensor(as.matrix(features), dtype = torch\_float32())  
labels = torch\_tensor(labels, dtype = torch\_float32())  
  
net = nn\_module(  
 initialize = function() {  
 self$hidden1 = nn\_linear(3, 10)  
 self$output = nn\_linear(10, 1)  
 },  
 forward = function(x) {  
 x %>% self$hidden1() %>% nnf\_relu() %>%  
 self$output() %>% nnf\_sigmoid()  
 }  
)  
model = net()  
  
loss\_fn = nnf\_binary\_cross\_entropy  
optimizer = optim\_adam(model$parameters)  
  
for (epoch in 1:10) {  
 optimizer$zero\_grad()  
 output = model(features)  
 loss = loss\_fn(output, labels)  
 loss$backward()  
 optimizer$step()  
 cat("Epoch:", epoch, "Loss:", as.numeric(loss$item()), "\n")  
}

## Epoch: 1 Loss: 96.67   
## Epoch: 2 Loss: 96.67   
## Epoch: 3 Loss: 96.67   
## Epoch: 4 Loss: 96.67   
## Epoch: 5 Loss: 96.67   
## Epoch: 6 Loss: 96.67   
## Epoch: 7 Loss: 96.67   
## Epoch: 8 Loss: 96.67   
## Epoch: 9 Loss: 96.67   
## Epoch: 10 Loss: 96.67

predictions = model(features)  
predicted\_labels = predictions > 0.5  
  
predicted\_labels\_float = as.numeric(predicted\_labels)  
labels\_float = as.numeric(labels)  
  
# Calculate accuracy  
accuracy = mean(predicted\_labels\_float == labels\_float)  
print(paste("Accuracy:", accuracy))

## [1] "Accuracy: 0.0333"

Logistic Reg:

logistic\_model = glm(default ~ balance + income + student, data = Default, family = binomial)  
logistic\_predictions = predict(logistic\_model, type = "response")  
  
predicted\_labels\_logistic = ifelse(logistic\_predictions > 0.5, 1, 0)  
  
# Calculate accuracy  
accuracy\_logistic = mean(predicted\_labels\_logistic == Default$default)  
print(paste("Logistic Regression Accuracy:", accuracy\_logistic))

## [1] "Logistic Regression Accuracy: 0.9732"

## Question 10: In Section 10.9.6, we showed how to fit a linear AR model to the NYSE data using the lm() function. However, we also mentioned that we can “flatten” the short sequences produces for the RNN model in order to fit a linear AR model. Use this latter approach to fit a linear AR model to the NYSE data. Compare the test of this linear AR model to that of the linear AR model that we fit in the lab. What are the advantages/disadvantages of this approach.

library(torch)  
data(Smarket)  
nyse = Smarket  
nyse = nyse %>% mutate(log\_volume = log(Volume))  
  
# Convert data to torch tensors  
n = nrow(nyse)  
xrnn = as.matrix(nyse[, -1])  
xrnn = array(xrnn, c(n, 3, 5))  
xrnn = xrnn[,, 5:1]  
xrnn = aperm(xrnn, c(1,3,2))  
xrnn = apply(xrnn, 2, function(x) as.numeric(as.character(x)))  
xrnn = torch\_tensor(as.array(xrnn), dtype = torch\_float32())  
y = torch\_tensor(nyse[, "log\_volume"], dtype = torch\_float32())  
  
xrnn = xrnn %>%  
 torch\_reshape(c(prod(dim(xrnn))/15, 15))  
  
n\_rows = dim(xrnn)[1]  
set.seed(123)   
istrain = sample(c(TRUE, FALSE), n\_rows, replace = TRUE, prob = c(0.8, 0.2))  
  
# Define the model  
model = nn\_module(  
 initialize = function() {  
 self$flatten = nn\_flatten()  
 self$dense = nn\_linear(15, 1)  
 },  
 forward = function(x) {  
 x %>%   
 self$flatten() %>%  
 self$dense()  
 }  
)  
  
model = model()  
loss\_fn = nn\_mse\_loss()  
optimizer = optim\_rmsprop(model$parameters)  
  
if(length(istrain) != dim(xrnn)[1]) {  
 stop("Length of istrain does not match number of rows in xrnn")  
}  
  
  
  
for (epoch in 1:200) {  
 model$train()  
  
 xrnn\_train = xrnn[istrain, ]  
 y\_train = y[istrain]  
  
 preds = model(xrnn\_train)  
 preds\_squeezed = preds$squeeze()  
 loss = loss\_fn(preds\_squeezed, y\_train)  
 optimizer$zero\_grad()  
 loss$backward()  
 optimizer$step()  
}  
  
model$eval()  
npred = model(xrnn[!istrain, ])  
y\_test = y[!istrain]  
V0 = torch\_var(y\_test) \* (y\_test$numel() - 1)  
  
  
test.Rsquared = 1 - as.numeric(mean((y[!istrain] - npred)^2) / V0)  
test.Rsquared

## [1] NaN

This is returning nan, and I’m not exactly sure what’s going wrong.