We explored how to use embeddings to represent categorical variables. Furthermore, showed how to represent categorical variables with embeddings and add other variable to create a more complex model. Both posts focused on the Keras (R) functionality. Concluded that it feels artificial to represent categorical variables with embeddings in Keras. Especially concatenating multiple input layers is quite cumbersome with the current Keras interface.

Exploring Embeddings for Categorical Variables with Keras:

**Exploring Embeddings for Categorical Variables with Keras**

In order to stay up to date, I try to follow Jeremy Howard on a regular basis. In one of his recent videos, he shows how to use embeddings for categorical variables (e.g. weekdays).

First off; what are embeddings? An embedding is a mapping of a categorical vector in a continuous n-dimensional space. The idea is to represent a categorical representation with n-continuous variables. To make it more concrete, let’s say you want to model the effect of day of the week on an outcome. Usually you would try to one-hot encode the variable, which means that you create 6 variables (each for one day of a week minus 1) and set the variable 1 or 0 depending on the value. You end up having a 6-dimensional space to represent a weekday.  
So, what is the advantage of mapping the variables in an continuous space? In a nutshell; with embeddings you can reduce the dimensionality of your feature space which should reduce overfitting in prediction problems.

In order to test the idea on a play example, I downloaded the nyc citi bike count data from Kaggle. It contains daily bicycle counts for major bridges in NYC.

#https://www.kaggle.com/new-york-city/nyc-east-river-bicycle-crossings

df <- read.csv("data/nyc-east-river-bicycle-counts.csv")

df$date <- as.Date(df$Date)

df$weekday <- lubridate::wday(df$date)

df$users <- df$Brooklyn.Bridge

df <- df[df$users>0,]

df <- df[!is.na(df$users),]

df <- df[!is.na(df$weekday),]

df$ScaledUsers <- scale(df$users)

Next, we set up a sequentual model with keras. The first layer is the embedding layer with the size of 7 weekdays plus 1 (for the unknowns). The embedding-size defines the dimensionality in which we map the categorical variables. Jeremy Howard provides the following rule of thumb; embedding size = min(50, number of categories/2).

require(keras)

embedding\_size <- 3

model <- keras\_model\_sequential()

model %>% layer\_embedding(input\_dim = 7+1, output\_dim = embedding\_size, input\_length = 1, name="embedding") %>%

layer\_flatten() %>%

layer\_dense(units=40, activation = "relu") %>%

layer\_dense(units=10, activation = "relu") %>%

layer\_dense(units=1)

model %>% compile(loss = "mse", optimizer = "sgd", metric="accuracy")

hist <- model %>% fit(x = as.matrix(df$weekday), y= as.matrix(df$ScaledUsers), epochs = 50, batch\_size = 2)

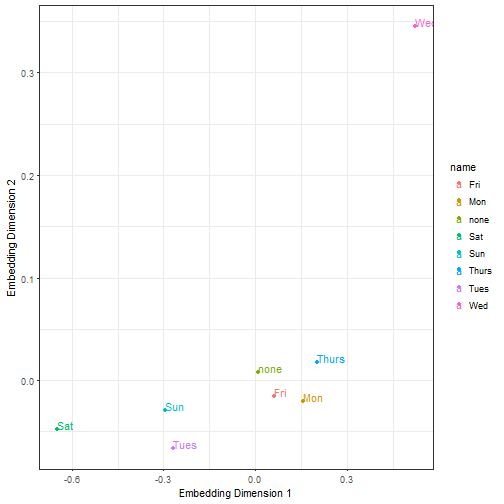
It is important to have a rather small batch size and to scale the count data. After training the model, we can extract individual layers. We named the first layer “embedding”. The weights of the embedding layer define where in the 3-dimensional feature space the network has placed the variables. We can show that by plotting the points in a scatterplot.

layer <- get\_layer(model, "embedding")

embeddings <- data.frame(layer$get\_weights()[[1]])

embeddings$name <- c("none", levels(wday(df$date, label = T)) )

ggplot(embeddings, aes(X1, X2, color=name))+ geom\_point() +geom\_text(aes(label=name),hjust=0, vjust=0) + theme\_bw() + xlab("Embedding Dimension 1") +ylab("Embedding Dimension 2")



The great thing about the embedding layer weights are, that they act as a lookup table. Merging the variables back to our dataset we can use the dimensions as input (X1, X2, X3) for a simple linear regression replacing the categorical representation of the day of the week variable.

df$weekDayF <- wday(df$date, label = T)

embeddings$lookup <- c("none", levels(df$weekDayF))

dff <- merge(df, embeddings, by.x="weekDayF", "lookup")

## we trained the embeddings on the Brooklyn.Bridge variable but test it on another ...

dff$users <- dff$Manhattan.Bridge

testRun <- function(x){

sample <- caret::createDataPartition(dff$weekDayF, list=FALSE, p = 0.8)

train <- dff[sample,]

test <- dff[-sample,]

fit1 <- lm(users ~ X1 + X2 + X3, data=train)

fit2 <- lm(users ~ weekDayF , data=train) # 6 input variables as lm automatically one-hot encodes categorical variables.

data.frame(run=x,

Embedding=sqrt(mean((predict(fit1, test) - test$users)^2)),

Categorical=sqrt(mean((predict(fit2, test) - test$users)^2) ))

}

test <- plyr::ldply(1:100, function(x){testRun(x)})

mm <- data.table::melt(test, id.vars="run")

dd <- plyr::ddply(mm,.(variable), summarise, m=mean(value))

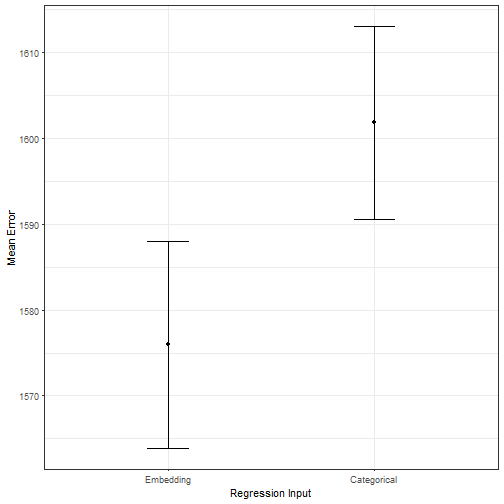
df2 <- Rmisc::summarySE(mm, "value", "variable")

ggplot(df2, aes(x=variable, y=value)) +

geom\_point()+

geom\_errorbar(aes(ymin=value-se, ymax=value+se), width=.2,

position=position\_dodge(0.05)) +ylab("Mean Error") + xlab("Regression Input") + theme\_bw()



Further, we can test if the embedding model outperforms the categorical regression model in an out of sample evaluation. The plot above shows the mean error over 100 test-runs (training on 80%, testing on 20%, metric: RMSE.) The embedding model (with a more compact representation of the day of the week) outperforms the categorical model.

I watched the official release video about the tfestimators package. It turns out that the awesome rstudio team build a very handy interface to access tensorflow and train models with multiple parameters and embeddings.

In this post, I will show how to use the package to quickly fit a model in which categorical variables are represented as embeddings.

As in the posts before, I work with the nyc citi bike count data from Kaggle. It contains daily bicycle counts for 4 major bridges in NYC. In order to have a longer dataset, I use the bicycle count for all bridges as the dependent variable.

#<https://www.kaggle.com/new-york-city/nyc-east-river-bicycle-crossings>

df <- read.csv("data/nyc-east-river-bicycle-counts.csv")

dflong <- data.table::melt(df[c("Date", "Brooklyn.Bridge", "Manhattan.Bridge", "Williamsburg.Bridge" ,"Queensboro.Bridge")], idvars="date")

dflong$date <- as.Date(dflong$Date)

dflong$weekday <- wday(dflong$date, label = T)

dflong <- merge(dflong, df[, c("Date", "Precipitation", "Low.Temp..Â.F.")], by="Date")

dflong$ScaledUsers <- scale(dflong$value)

dflong$lowTemp <- scale(dflong[,"Low.Temp..Â.F."])

dflong$rain <- ifelse(dflong$Precipitation != 0, 0,1)

dflong$Bridge <- factor(dflong$variable)

levels(dflong$Bridge) <- 1:length(levels(dflong$Bridge))

levels(dflong$weekday) <- 1:length(levels(dflong$weekday))

The goal of our play model is to predict the number of bicycle per day on a certain bridge dependent on the weekday, the bridge (“Brooklyn.Bridge”, “Manhattan.Bridge”, “Williamsburg.Bridge” ,”Queensboro.Bridge”), if it rains and the temperature. So overall we have 2 categorical variables, one binary and one continuous variable.

library(tfestimators)

## convert the factor to integer -- tfestimators is strict with input types.

dflong$Bridge <- as.integer(dflong$Bridge)

dflong$weekday <- as.integer(dflong$weekday)

embedding\_dimension\_bridges = 2

embedding\_dimension\_weekdays = 3

cols <- feature\_columns(

column\_numeric("lowTemp","rain"),

column\_embedding(column\_categorical\_with\_vocabulary\_list("weekday", vocabulary\_list = c(1:7)), embedding\_dimension\_weekdays),

column\_embedding(column\_categorical\_with\_vocabulary\_list("Bridge",vocabulary\_list = c(1:4)), embedding\_dimension\_bridges)

)

The first step that we need to do is to define the input variables and their type. Let’s start with the simple numeric variables lowTemp and rain. We just define the input as “column\_numeric(“lowTemp”,”rain”)”. Next, the two categorical variables that we want to embed, need a bit more work. a) they need a list of all possible values (defined in vocabulary\_list parameter). Additionally, we need to define the embedding\_dimension for each categorical variable.

Next, we write a short function that defines the input and output of the the model as well as batch size and number of epochs.

library(tfestimators)

bridge\_input\_fn <- function(data, num\_epochs = 1) {

tfestimators::input\_fn(data,

features = c("lowTemp","rain","weekday", "Bridge"),

response = "ScaledUsers",

batch\_size = 2,

num\_epochs = num\_epochs)

}

############ Train and Test Dataset #############

indices <- sample(1:nrow(dflong), size = 0.80 \* nrow(dflong))

train <- dflong[indices, ]

test <- dflong[-indices, ]

############## Define the model ############

model <- dnn\_regressor(feature\_columns = cols, hidden\_units = c(32, 10), dropout = 0.15)

# train the model

history <- model %>% train(bridge\_input\_fn(train[,c("ScaledUsers", "lowTemp","rain","weekday", "Bridge" )], num\_epochs = 1))

## eval the output

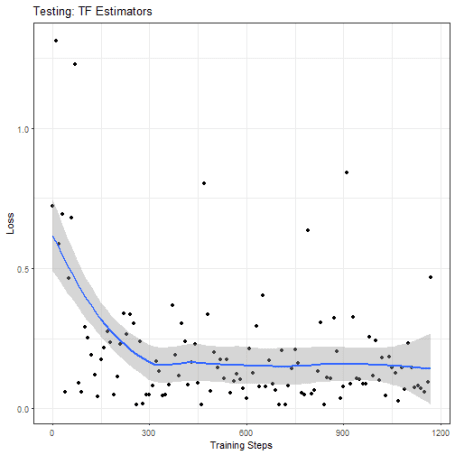
model %>% evaluate(bridge\_input\_fn(test))

In order to evaluate the model, we split the data in train and test set. We define the model as deep neural network (DNN), regression model with two hidden layers (one with 32 the other with 10 nodes). Compared to the Keras version, in which one needs to concatenate different input layers this interface is straight-forward.  
Finally, we check the model’s accuracy on test set and print the learning history.

require(ggplot2)

df <- data.frame(losses = history$losses$mean\_losses, steps=history$step)

ggplot(df, aes(steps, losses))+ geom\_point() +geom\_smooth() +theme\_bw() + ylab("Loss") + xlab("Training Steps") + ggtitle("Testing: TF Estimators")



To conclude, the package is a great step forward to apply deep neural nets to everyday problems and to quickly use embeddings for categorical variables.