Drought forecasting exercise

Jordan Richards

2023-01-28

Dataset

- Exercise developed by Chris Wikle and Dan Pagendam (2019).
- The data consists of:
  - monthly 33 x 84 grids (2 degree x 2 degree) of sea surface temperature (SST) anomaly (2772 pixels).
  - monthly rainfall anomaly in mm for the Murray Darling Basin (MDB).
- Obtained from two sources:
  - http://www.bom.gov.au/climate/change/
  - http://iridl.ldeo.columbia.edu/
- We will use a type of recurrent NN(LSTM) model to obtain 3-month-out forecasts of rainfall anomaly using SST grids as a predictor.

Required packages

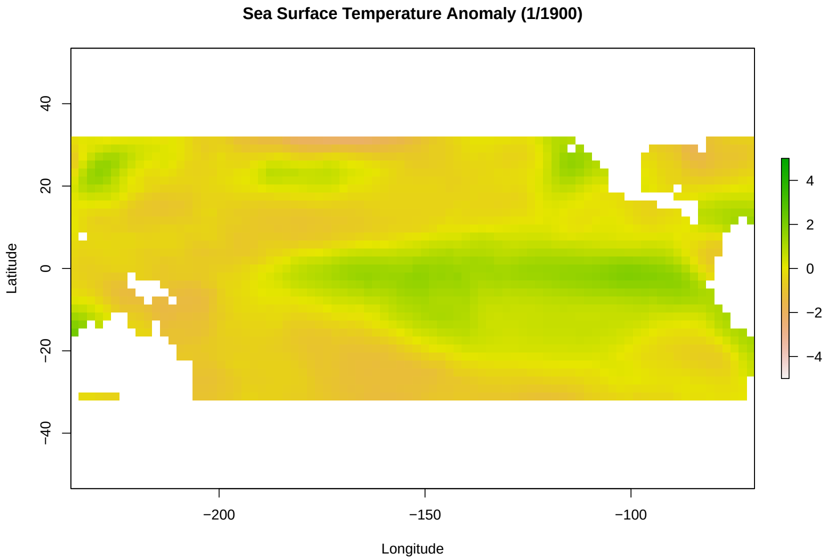
We will be using some functions and the images from Dan's github directory, https://github.com/dpagendam/deepLearningRshort

```
#remotes::install_github("dpagendam/deepLearningRshort")

library(keras)
library(raster)
```

```
## Loading required package: sp
```

```
library(deepLearningRshort)
data(drought)
```

Visualise SST



Strategy

- We could apply a CNN layer recurrently to extract both sequential and spatial information from the SST grids, but this will require lots of processing power and parameters
- Instead, we treat the SST as a multivariate time series and use regular RNNs. However, we have 2772 locations, so we first reduce the dimensionality using EOFs (PCA)

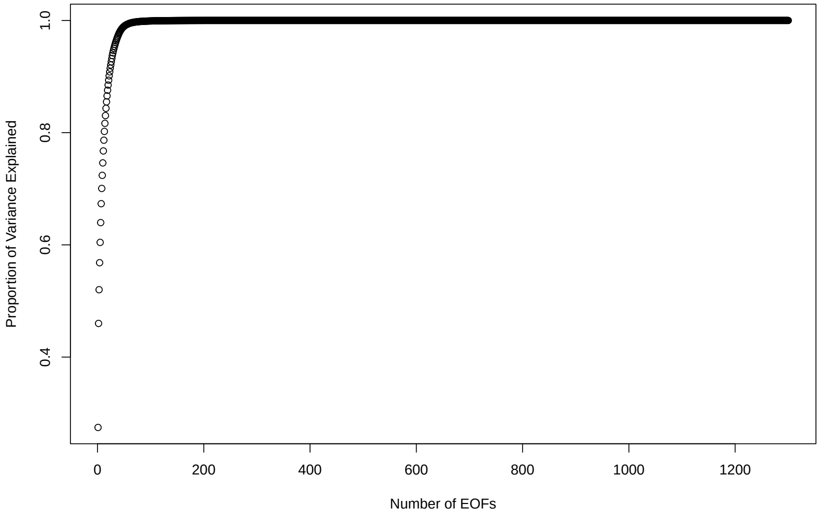
Data manipulation

```r
batchSize <- 32
forecastMonthsAhead <- 3
timestepsPerSample <- 24
trainingInds <- 1:1300
validationInds <- 1301:1434
# We consider only 1434 months
```

- We will project the 2772 pixels onto 100 EOFs.

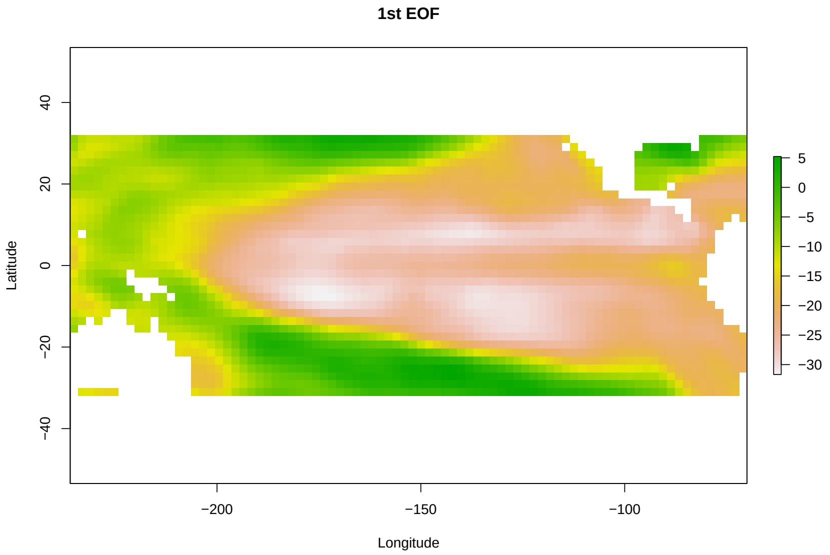
```r
numComponents <- 100
EOFList <- rasterToEOFs(anomalyRasterList[trainingInds],
numComponents = numComponents, plot = FALSE)
v.train <- EOFList[["rasterEOFs"]][["v.dim.red"]]
```

Plot EOFs

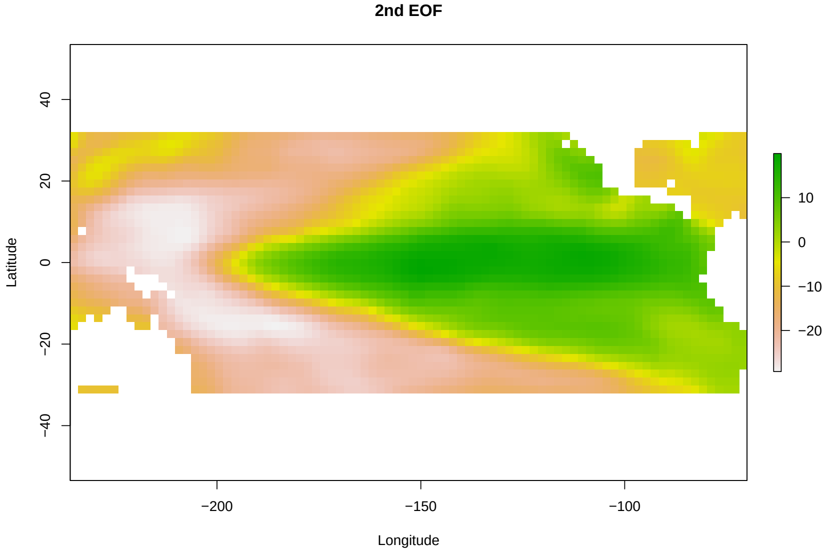


Proportion of Variance Explained
0.4
0.6
0.8
1.0
0
200
400
600
800
1000
1200
Number of EOFs

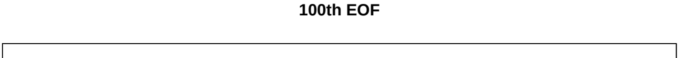
Plot EOFs


1st EOF
Latitude
40
20
0
−20
−40
Longitude
−200
−150
−100
5
0
−5
−10
−15
−20
−25
−30

Plot EOFs



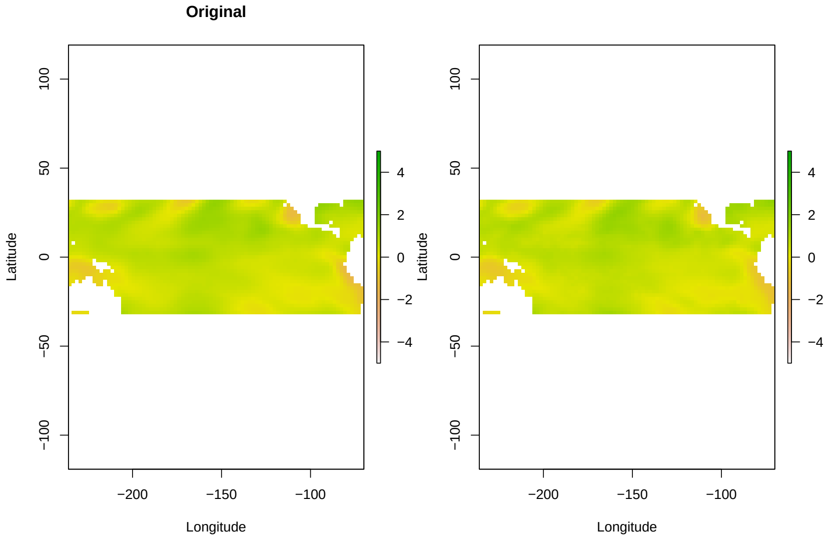
Plot EOFs



Reconstructing SST

```r
validationSample <- 1434
X <- EOFList$rasterEOFs$EOFs
r1 <- anomalyRasterList[[validationSample]]
validPixels <- EOFList[["raster.validPixels"]]
Y <- getValues(r1)
Y <- Y[validPixels]
lm1 <- lm(Y~X)
intercept <- coefficients(lm1)[1]
alpha <- coefficients(lm1)[1]
beta <- coefficients(lm1)[2:(numComponents + 1)]
r2 <- alpha + EOFsToRaster(X, matrix(beta, nrow = 1),
      c(33, 84), validPixels)[[1]]
extent(r2) <- extent(r1)
```

Reconstructing SST

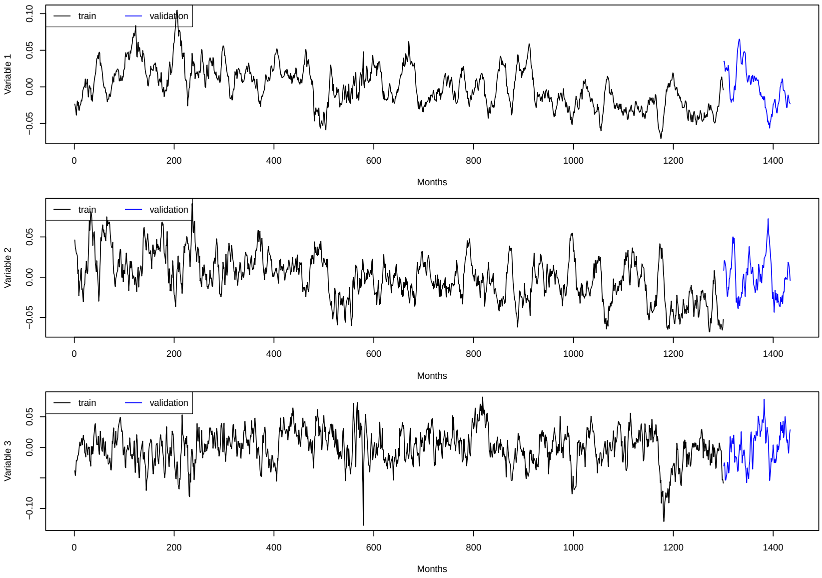


Dimension reduction for validation data

- Here we project the validation data SST anomaly grids onto the same EOFs generated from the training data. · You can think of v.validation as a multivariate time series of coefficients that we can use to reconstruct SST anomaly from the EOFs.

```
v.validation <- proj.raster.EOFs(
  anomalyRasterList[validationInds],
          EOFList[["rasterEOFs"]][["EOFs"]],
          EOFList[["raster.validPixels"]])
```

Dimension reduction for validation data



Data wrangling

- All predictors combined together

```
v.combined <- rbind(v.train, v.validation)
```

- and normalised

```
v.scaling.train <- scaleCols.pos(
  v.combined[trainingInds, ])
v.train.scaled <- v.scaling.train[["X.scaled"]]
v.scaling.validation <- scaleCols.pos(
  v.combined[validationInds, ],
              colMaxsX = v.scaling.train[["colMaxsX"]],
              colMinsX = v.scaling.train[["colMinsX"]])
v.validation.scaled <- v.scaling.validation[["X.scaled"]]

v.scaled <- rbind(v.train.scaled, v.validation.scaled)
```

Formatting data for an RNN

```
numDims <- ncol(v.scaled)
tensorData <- tensorfyData.rnn(v.scaled,
                forecastMonthsAhead,
                timestepsPerSample, indicesX = 1:numDims,
                indicesY = 1:numComponents,
                indicesTrain = trainingInds,
                indicesTest = validationInds)
str(tensorData)
```

```
## List of 8
##  $ X.train.rnn   : num [1:1273, 1:24, 1:100] 0.268 0.248
##  $ Y.train.rnn   : num [1:1273, 1:100] 0.396 0.323 0.363
##  $ X.test.rnn    : num [1:107, 1:24, 1:100] 0.597 0.602
##  $ Y.test.rnn    : num [1:107, 1:100] 0.58 0.654 0.7 0.7
##  $ x.train.tsInds: int [1:1273] 24 25 26 27 28 29 30 31
##  $ x.test.tsInds : int [1:107] 1324 1325 1326 1327 1328
##  $ y.train.tsInds: int [1:1273] 27 28 29 30 31 32 33 34
##  $ y.test.tsInds : int [1:107] 1327 1328 1329 1330 1331
```

Response data

```
Y.train.inds <- tensorData$y.train.tsInds
Y.valid.inds <- tensorData$y.test.tsInds
Y.train.rnn_MDB <- rainfallAnomaly[Y.train.inds, 3]
Y.valid.rnn_MDB <- rainfallAnomaly[Y.valid.inds, 3]
```

- We will also normalise the response, but this is only because we will fit a Gaussian model

```
Y.train.min <- min(Y.train.rnn_MDB)
Y.train.max <- max(Y.train.rnn_MDB)

Y.rnn.train <- (Y.train.rnn_MDB - Y.train.min)/
                    (Y.train.max - Y.train.min)
Y.rnn.valid <- (Y.valid.rnn_MDB - Y.train.min)/
                    (Y.train.max - Y.train.min)
```

RNN tensors

We finally get

```
X.rnn.train <- tensorData[["X.train.rnn"]]
X.rnn.valid <- tensorData[["X.test.rnn"]]

dim(X.rnn.train)
```

```
## [1] 1273   24  100
```

```
length(Y.rnn.train)
```

```
## [1] 1273
```

Custom loss

```
Gaussian_logLikelihood <- function(y_true, y_pred)
{
  K <- backend()
  # Extract the first and second columns of predictions
  mu <- (y_pred[,1])
  sigma <- K$exp(y_pred[,2])

#Extract first column of y_true to ensure same dimension
  y <- y_true[,1]

  ll <- -0.5*((mu - y)/(sigma))^2 - K$log(sigma)
  ll <- ll -0.5*K$log(2*pi)

  return( -(K$sum(ll)))
}
```

Building an LSTM Model

```
library(keras)
input.lay <- layer_input(shape = dim(X.rnn.train)[-1],
      name = 'input_layer')

model <- input.lay %>%
  layer_lstm(units = 128,
      input_shape = c(timestepsPerSample, numDims),
      return_sequences = FALSE) %>%
  layer_dense(units = 128, activation = "relu")
  output.lay <- model %>% layer_dense(units = 2)
```

Compile

```
  model <- keras_model(
    inputs = c(input.lay),
    outputs = c(output.lay)
  )
model %>% compile(loss = Gaussian_logLikelihood,
                  optimizer = optimizer_rmsprop())
```

Summary

```
summary(model)
```

Model training

Let's train the model with early stopping and a checkpoint

```
history <- model %>% fit(
  x = X.rnn.train, y = Y.rnn.train,
  batch_size = batchSize, epochs = 200, shuffle = FALSE,
  validation_data = list(X.rnn.valid, Y.rnn.valid),
  callbacks = list(
  callback_early_stopping(monitor = "val_loss",
                          min_delta = 0, patience = 20),
  callback_model_checkpoint(filepath = "model_weights",
      verbose=0, monitor="val_loss",
      save_best_only = TRUE, save_weights_only = TRUE)))
#Then load the saved weights
model <- load_model_weights_tf(model,
                               filepath="model_weights")
```

Checking performance

- We can calculate the mean and standard deviations of the 3 month out (Gaussian) predictive distributions.
- Then create 50% and 95% prediction intervals.

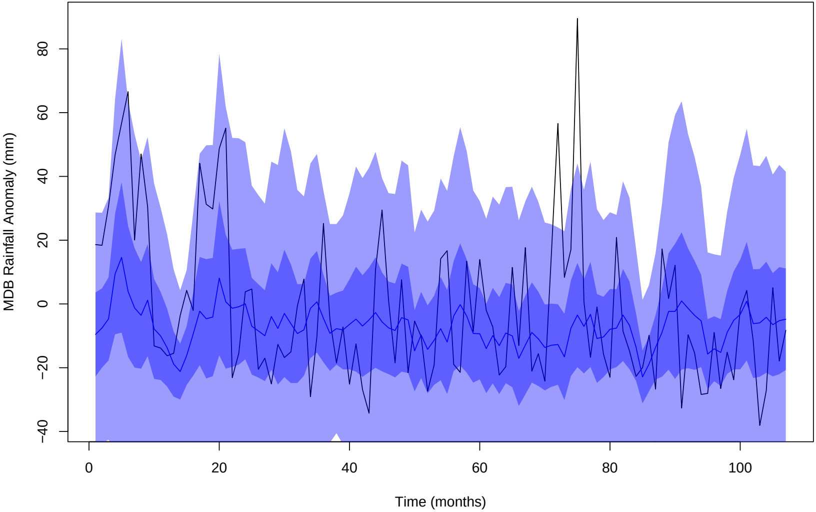
```
lstmPredictions <- model %>% predict(X.rnn.valid)

mu <- Y.train.min +
  lstmPredictions[, 1]*(Y.train.max - Y.train.min)
sigma <- exp(lstmPredictions[, 2])*
  (Y.train.max - Y.train.min)
n <- length(mu)
upper95 <- mu + 1.96*sigma
lower95 <- mu - 1.96*sigma
upper50 <- mu + 0.674*sigma
lower50 <- mu - 0.674*sigma
```

Checking performance

Plot the true time series with 3-month-out forecast and 50% and 95% prediction intervals.

```
plot(rainfallAnomaly[Y.validation.inds, 3], ty = "l",
     xlab = "Time (months)",
     ylab = "MDB Rainfall Anomaly (mm)")
lines(mu, col = "blue")
polygon(x = c(1:n, rev(1:n), 1),
        y = c(lower95, rev(upper95), lower95[1]),
        col = fade("blue", 100), border = NA)
polygon(x = c(1:n, rev(1:n), 1),
        y = c(lower50, rev(upper50), lower50[1]),
        col = fade("blue", 100), border = NA)
```

Checking performance



Checking performance

Calculate what percentage of the time the true rainfall anomaly was within the 50% and 95% prediction intervals.

```
n <- (length(Y.valid.inds))
coverage50 <- length(
  which(rainfallAnomaly[Y.valid.inds, 3]> lower50
  & rainfallAnomaly[Y.valid.inds, 3] < upper50))/n
coverage95 <- length(
  which(rainfallAnomaly[Y.valid.inds, 3] > lower95
  & rainfallAnomaly[Y.valid.inds, 3] < upper95))/n
print(coverage50)
```

```
## [1] 0.5514019
```

```
print(coverage95)
```

```
## [1] 0.9626168
```

Extensions

- How does varying the number of units in the LSTM layer affect the predictions?
- How do the predictions change if you add three dense layers after the LSTM layer (instead of just 1)?
- How are the predictions if much fewer EOFs are used for prediction?
- How does fewer EOFs affect the number of parameters in the model?