Arificial Neural Networks

Modelling Corn Bunting's habitat suitability

Francesco Maria Sabatini

26 June 2020

Contents

1 Goal of the seminar

- to understand the concept of artificial neural networks (in this case particularly the multilayer perceptron)
- to analyse a comprehensive data set of various meteorological variables and structural parameters (measurements) representing the habitat suitability of the bird family "corn bunting" (Emberiza calandra) with the help of artificial neural networks
- to derive an appropriate prediction model for analyzing and interpreting the given value
- to interpret the resulting analysis and prediction results

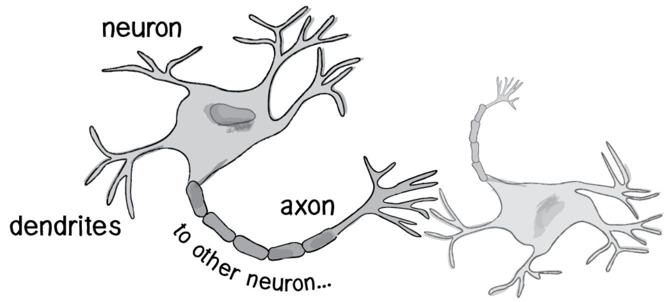
if time allows:

- to visualize prediction results in a spatial context with geographic coordinates

Analysis will be performed in R.

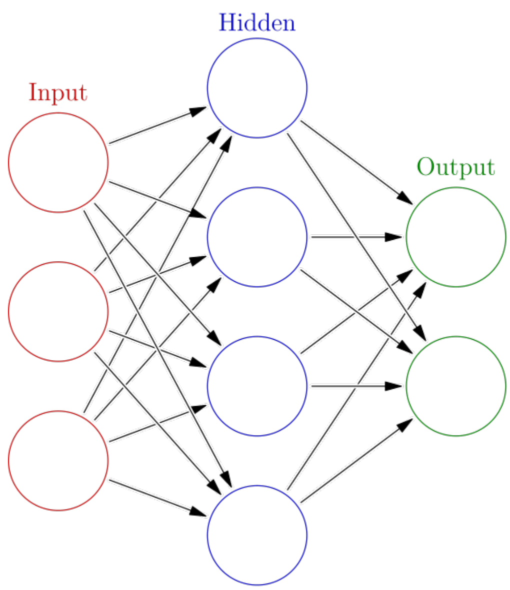
2 Artificial Neural Networks (ANN)

Artificial neural networks are a set of algorithms loosely inspired to the human brain. They are great at recognizing patterns, and require limited upfront specifications. Similarly to the human brain, they are composed of networks of units (=neurons; =nodes). Each node receives information from either the outer world, or from those nodes located upstream the feed of information. Each node processes the information independently, and returns a result, which can either be recombined into an output, or fed to the nodes downstream.



Schematic representation of a human neuron. Source: Natureofcode.com

Neural networks are organized in **layers**, whose nodes are normally fully-connected to all the nodes of the upstream and downstream layers (but not within the same layer). Layers can have a different number of nodes. The last layer is called 'output' layer. All upstream layers are called 'hidden' layers. Inputs are not normally considered an independent layer. Nodes are also known as **Perceptrons** from the pioneer work of Frank Rosenblatt.



Example of fully-connected two-layer Artificial Neural Network. Source: Wikipedia.com

Multilayer neural networks are extremely flexible, and are considered universal: i.e., with the right number of layers, and nodes per layer, a multilayer NN can approximate with arbitrary accuracy any realistic function. This is stated by the **Universality Theorem**. Yet this theorem doesn't say:

- how many hidden neurons, nor
- how many layers

should be there

In other words a solution does exist, yet there is no guarantee we will ever find it.

2.1 Training a neural network

Neural networks need to be trained. This means that we need to provide a batch of **train data**, for which there is a known answer. In this way, the network can find out if it has made correct guesses. If a guess is incorrect, the network can learn from its mistake and adjust itself. This method is called **supervised learning**.

Training a neural networks goes in iterative steps:

1. Inputs flow through the nodes, where they are processed as **weighted sums**
2. An **activation function** transforms the weighted sum into a numerical result which is **propagated forward** into the nodes of the next layer(s), and eventually, to the output
3. The output is then compared to the known answer, and the **error** calculated
4. Weights are adjusted based on the error through a process called **Error backpropagation**
5. Repeat steps 1-4 several (thousands!) of times.

Once the neural network is trained, we can use **test data** to evaluate its performance. Test data are data for which there is a known answer, but were not used during the training process. Comparing the predictions to the known answers, allows to calculate the overall performance of the ANN.

If we are happy about the performance of our NN, we can finally use it on real world data, i.e, we can use it to make predictions based on data whose answers are not know.

3 Modelling habitat suitability of Corn Bunting with ANN

The corn bunting (*Emberiza calandra*) is a passerine bird in the bunting family Emberizidae. It breeds across southern and central Europe, north Africa and Asia across to Kazakhstan. It is mainly resident, but some birds from colder regions of central Europe and Asia migrate southwards in winter.

The corn bunting is a bird of open country with trees, such as farmland and weedy wasteland. It has declined greatly in north-west Europe due to intensive agricultural practices depriving it of its food supply of weed seeds and insects, the latter especially vital when feeding the young. (Source Wikipedia.com)

Illustration of Corn Bunting. Source: Wikipedia.com

Here, we will use ANN to train a model predicting the occurrence of the Corn Bunting over a landscape in

Brandenburg.

3.1 Prepare and check data

Let's open up R

```
# Install packages, if needed
#install.packages(c("dplyr", "readxl", "sf", "ggplot2",
#                        "neuralnet", "corrplot", "cowplot"))

#Load Required libraries
# improved data manipulation
library(dplyr)
library(corrplot)
library(readxl)
# Spatial
library(sf)
# visualization
library(ggplot2)
library(cowplot)
# neural networks
library(neuralnet)
```

We can now import our input data.

Data and code are temporally available at the following link:

https://portal.idiv.de/nextcloud/index.php/s/qkEELH8ywtJnYAW

```
## import data
Biotopes_sf <- st_read("data/Brandenburg_Biotops_2009.gpkg")
grid_sf <- st_read("data/zt_v100_UTM33N.gpkg")
mydata <- read_xls("data/zt_habitat_corn_bunting.xls",sheet = 1)
```

Let's check the data:

```
head(mydata)
```

```
## # A tibble: 6 x 10
##   ZT_V100_ID    NS    TM    WN    SN    AK    DL    WW   HSI class
##        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1          2   635  8.32   100  1655     0     0     0     0     0
## 2          3   635  8.32     0  1615     0     0     0     0     0
## 3          4   635  8.32     0  1581     0     0     0     0     0
## 4          5   635  8.32     0  1552     0     0     0     0     0
## 5          6   630  8.37     0  1529     0     0     0     0     0
## 6          7   630  8.37     0  1513     0     0     0     0     0
```

The dataframe `mydata` contains 16675 rows, and 10 variables. The naming is a bit obscure. Here's a legend:

- zt_v100_ID: index of fishnet cell within fishnet zt_v100_gk5.shp
- ns (Jahres-Niederschlagssumme): annual sum of precipitation in [mm]
- tm (Jahres-Durchschnittstemperatur): annual temperature mean in [°C]
- wn (Waldnähe): distance to forest in [m]
- sn (Siedlungsnähe): distance to settlements in [m]
- ak (Fläche der Ackerkulturen): area of arable crops within fishnet cell [0 – 10,000 m^2]
- dl (Fläche lehmiger Böden): area of loamy soils within fishnet cell [0 – 10,000 m^2]: the share of loamy soils may be considered as indicator for the annual and perennial vegetation

- ww (Fläche von Wiesen und Weiden): area of pastures and meadows within fishnet cell [0 – 10,000 m^2]
- HSI (Habitat Suitability Index): Habitat Suitability Index evaluated by experienced ornithologists; HSI takes values from 0 (unsuitable habitat) till 1 (optimal habitat)
- class (classified Habitat Suitability Index): expert split of HSI into two classes (class 0 = rather unsuitable habitat; class 1 = rather suitable habitat)

Please note that the column `zt_v100_ID` corresponds to the index of the 100 x 100 m fishnet used in the object `grid`. Also note that the last column (`class`), is simply a discretization of the column `HSI`. This will be our response variable when training the network.

How are these variables distributed?

```r
summary(mydata)
```

```
##    ZT_V100_ID          NS              TM              WN
##  Min.   :    1   Min.   :583.0   Min.   :8.090   Min.   :   0.0
##  1st Qu.: 4170   1st Qu.:598.0   1st Qu.:8.320   1st Qu.:   0.0
##  Median : 8338   Median :614.0   Median :8.390   Median : 200.0
##  Mean   : 8338   Mean   :615.2   Mean   :8.386   Mean   : 272.1
##  3rd Qu.:12506   3rd Qu.:630.0   3rd Qu.:8.460   3rd Qu.: 412.0
##  Max.   :16675   Max.   :677.0   Max.   :8.600   Max.   :1627.0
##        SN               AK              DL              WW
##  Min.   :   0.0   Min.   :    0   Min.   :    0   Min.   :    0
##  1st Qu.: 412.0   1st Qu.:    0   1st Qu.:    0   1st Qu.:    0
##  Median : 728.0   Median :    0   Median :  953   Median :    0
##  Mean   : 800.4   Mean   : 3171   Mean   : 3014   Mean   : 1423
##  3rd Qu.:1118.0   3rd Qu.: 7955   3rd Qu.: 6022   3rd Qu.:  298
##  Max.   :2961.0   Max.   :10000   Max.   :10000   Max.   :10000
##       HSI             class
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.1700   Median :0.0000
##  Mean   :0.2208   Mean   :0.2318
##  3rd Qu.:0.3300   3rd Qu.:0.0000
##  Max.   :0.7900   Max.   :1.0000
```

```r
table(mydata$class)
```

```
##
##     0     1
## 12809  3866
```

Our response variable spans between 0 (unsuitable) and 1 (suitable), which is fine. The predictors, instead, have wildly different ranges. `NS` for instance ranges between 0 and 677, while other variables (e.g., `AK`) range between 0-10000. This can be a problem when training our neural network. It is highly recommended to standardize all predictors before training a NN. We do it now.

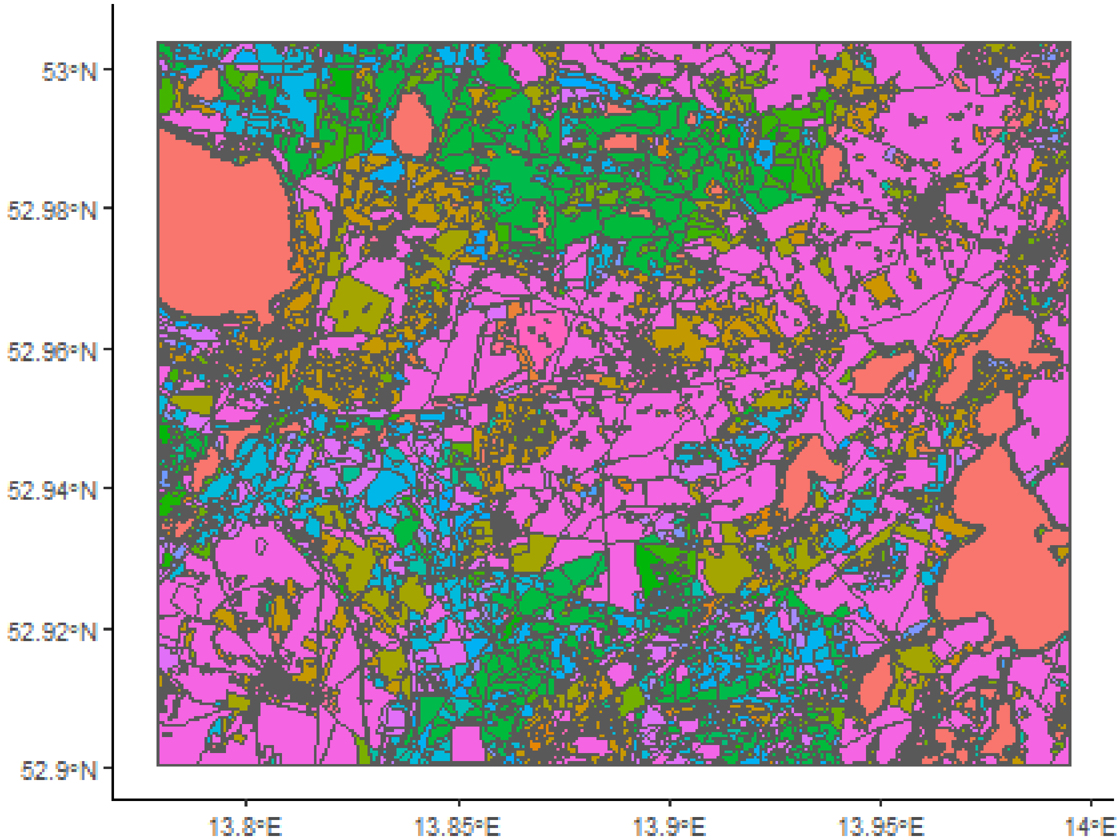
```r
## standardize variables to 0 - 1 range
# we drop the first (index) and the last two(HSI and response variable)columns
predictors <- mydata[,c(2:8)]
maxs <- apply(predictors, 2, max)
mins <- apply(predictors, 2, min)

scaled <- as.data.frame(scale(predictors, center = mins, scale = maxs - mins))
```

```r
mydata.scaled <- data.frame(class=mydata$class, scaled)
summary(mydata.scaled)
```

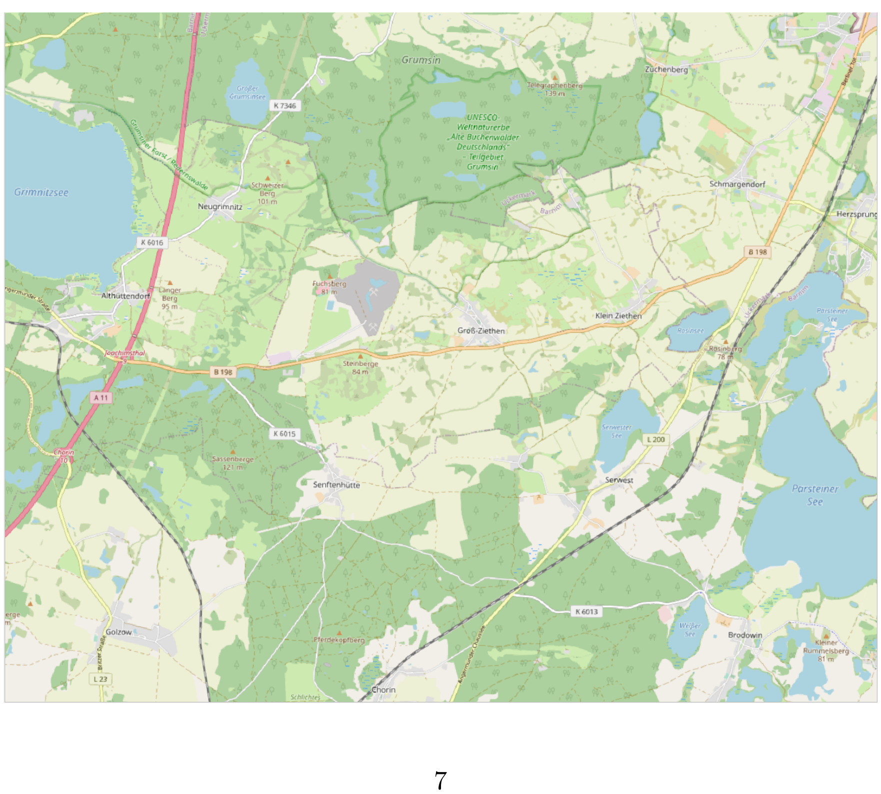
```
##      class              NS               TM               WN        
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.0000   1st Qu.:0.1596   1st Qu.:0.4510   1st Qu.:0.0000  
##  Median :0.0000   Median :0.3298   Median :0.5882   Median :0.1229  
##  Mean   :0.2318   Mean   :0.3427   Mean   :0.5803   Mean   :0.1672  
##  3rd Qu.:0.0000   3rd Qu.:0.5000   3rd Qu.:0.7255   3rd Qu.:0.2532  
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
##        SN               AK               DL               WW        
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.1391   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000  
##  Median :0.2459   Median :0.0000   Median :0.0953   Median :0.0000  
##  Mean   :0.2703   Mean   :0.3171   Mean   :0.3014   Mean   :0.1423  
##  3rd Qu.:0.3776   3rd Qu.:0.7955   3rd Qu.:0.6022   3rd Qu.:0.0298  
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
```

Now, let's take a quick look to the spatial data. If you are unfamiliar with the plotting package `ggplot2`, don't worry. Just take a look at the output. Any alternative way of visualizing the data would work equally good.

```r
#plot data
#Careful, the graph below renders slowly
(Landuse <- ggplot(data=Biotopes_sf) +  ## renders slowly
  geom_sf(aes(fill=Biotyp_8st)) + # use column biotyp_8st as a color code
  theme_classic() +
  theme(legend.position = "none") #+
  #geom_sf(data=grid_sf, fill=NA)
)
```



The land use data seem to have an appropriate spatial projection. The color coding is counter-intuitive, but the spatial data seem to be consistently defined. You can try adding the layer of the fishnet grid (commented in code) to check that the grid is properly aligned too.
Let’s compare this tile to a screenshot from OpenStreetMap



Study area. Source: OpenStreetMap

This helps us understand better the legend. We can clearly see a couple of big lakes (orange, Grimnitzsee to the West, Parsteinersee to the SE), and some infrastructure lines. The linear element cutting the NW corner of the tile is the highway A11. Purple loosely corresponds to agricultural land. Green and blue colors represent different kinds of forests.

Let's take a closer look at the content of the `Biotopes` dataset.

```
head(Biotopes_sf)
```

```
## Simple feature collection with 6 features and 19 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 417870.2 ymin: 5872853 xmax: 421086 ymax: 5873414
## projected CRS:  ETRS89 / UTM zone 33N
##   fk_verwalt  fk_tk Gebnra   id           pk_ident Fk_inten  FK_Biotyp
## 1   LU12010- 2948SO   0909  909 LU12010-2948SO0909        A      12630
## 2   LU12010- 2948SO   1278 1278 LU12010-2948SO1278        A    0510301
## 3   LU12010- 2948SO   1286 1286 LU12010-2948SO1286        A    0510301
## 4   LU12010- 2948SO   1415 1415 LU12010-2948SO1415        A      12263
## 5   LU12010- 2948SO   1265 1265 LU12010-2948SO1265        A 0832000093
## 6   LU12010- 2948SO   1280 1280 LU12010-2948SO1280        A      08103
##   Bemerkung Fk_bioalte        lubi_nr kart_date Shape_Leng Shape_Area
## 1      <NA>       <NA> DOP050_418-872  8.9.2009  8154.9090 103310.744
## 2      <NA>       <NA> DOP050_418-872  8.9.2009   883.3010  31727.114
## 3      <NA>       <NA> DOP050_413-872  8.9.2009  1790.7357  47216.675
## 4      <NA>       <NA> DOP050_418-872  8.9.2009   261.8741   3838.421
## 5      <NA>       <NA> DOP050_418-872  8.9.2009  1661.7523  95758.669
## 6      <NA>       <NA> DOP050_418-872  8.9.2009   970.2773  44693.994
##   Biotyp_8st
## 1   12630000
## 2   05103010
## 3   05103010
## 4   12263000
## 5   08320000
## 6   08103000
##
## 1                                                                                         Autobahnen
## 2 Feuchtwiesen n<U+FFFD>hrstoffreicher Standorte; weitgehend ohne spontanen Geh<U+FFFD>lzbewuchs (< 
## 3 Feuchtwiesen n<U+FFFD>hrstoffreicher Standorte; weitgehend ohne spontanen Geh<U+FFFD>lzbewuchs (< 
## 4                                   Wohn- und Mischgebiete, Einzel- und Reihenhausbebauung mit Waldbau
## 5                                                                                            Buchen
## 6                                                                                   Erlen-Bruchw<U+FFF
##     WK        WK_Text   MF      MF_Text                           geom
## 1 <NA>           <NA> <NA>         <NA> MULTIPOLYGON (((421086 5873...
## 2 <NA>           <NA> <NA>         <NA> MULTIPOLYGON (((418108 5873...
## 3 <NA>           <NA> <NA>         <NA> MULTIPOLYGON (((418007.5 58...
## 4 <NA>           <NA> <NA>         <NA> MULTIPOLYGON (((418218.4 58...
## 5    9 ungleichaltrig    3 gruppenweise MULTIPOLYGON (((420641.2 58...
## 6 <NA>           <NA> <NA>         <NA> MULTIPOLYGON (((420995 5873...
```

There is a lot of information that we don't probably need. Just take a look at the `Biotyp_8st` columns, which we used for visualization, and the corresponding legend in the `CIR_text` column. There seems to be a problem with the encoding of the latter, but let's not worry about that for now.

3.2 Training a multilayer ANN

To train our ANN, we need to prepare our *train* and *test* datasets. Normally, this is done splitting the input dataset in two chunks. Often, 80% is used for training, and 20% for testing. Training the ANN is computing intensive, though, and our time limited. To reduce training time, let's use only 1/5th of our dataset for now.

```
set.seed(899) # set a seed, so to make the resampling procedure reproducible
mydata.subset <- sample_frac(mydata.scaled, size=0.2)
#function sample_frac comes from dplyr package!

#split in train and test
n <- nrow(mydata.subset)
share.train <- 0.8
train.id <- sample(1:n, n*share.train, replace=F)
train = mydata.subset[train.id,]
test = mydata.subset[-train.id,]
```

We are almost ready to fit our first NN. Last thing to do, is to specify its formula and think of the number of layers, and nodes per layers we need. There's no golden rule, but some helpful guidance comes from practice. Most of the problems can actually be solved with only two layers. Also, there is a rule of thumb which suggests that the number of nodes should be smaller (about 2/3rd) than the number of predictors.
Let's start with a simple ANN having 2 layers (i.e., only 1 hidden layer), and 5 nodes.

```
myformula <- as.formula("class ~ NS + TM + WN + SN + AK + DL + WW")
```

```
set.seed(900)
nn_5 <- neuralnet(myformula, train, hidden=c(5), lifesign='minimal', threshold=0.01,
                  linear.output = F)
```

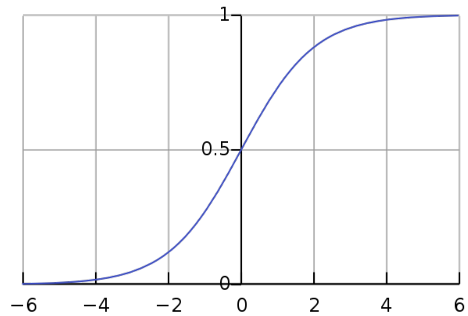
```
## hidden: 5    thresh: 0.01    rep: 1/1    steps: stepmax  min thresh: 0.0168149880998353
```

```
## Warning: Algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
```

Oooops!

Something went wrong and the training didn't succeed within the number of cycles (aka 'epochs') we preset. What to do now? We have different options We could increase the number of cycles, or decrease our sensitivity threshold. Check the function help `?neuralnet` and try to understand how this could be done. To understand what the sensitivity threshold means in practice, you can check this website: https://ml4a.github.io/ml4a/how_neural_networks_are_trained/.

Here is another useful trick, though. Part of the reason why our ANN is so slow at converging, depends on the fact that we are training it to return either zeros or ones. These values are at the extreme end of the activation function, which normally takes the shape of a sigmoid.



which is defined by the function:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

When the ANN is training, the errors are backpropagated to correct the weights. Now the problem is that the new weights are corrected of a quantity which is proportional to the derivative of the output of the sigmoid function. This derivative is equal to `y(1-y)`, where y is the output of the sigmoid. When y is close to 0 or 1, this quantity becomes extremely small, causing our ANN to converge extremely slowly.
That's why it can be helpful, sometimes, to recode our 0-1 response variables to new values, i.e., 0 -> 0.2, and 1 -> 0.8. Not convinced? Let's give it a try.

```
train_recoded <- train
train_recoded$class <- train_recoded$class*0.6+0.2
table(train_recoded$class)
```

```
##
##  0.2  0.8
## 2030  638
```

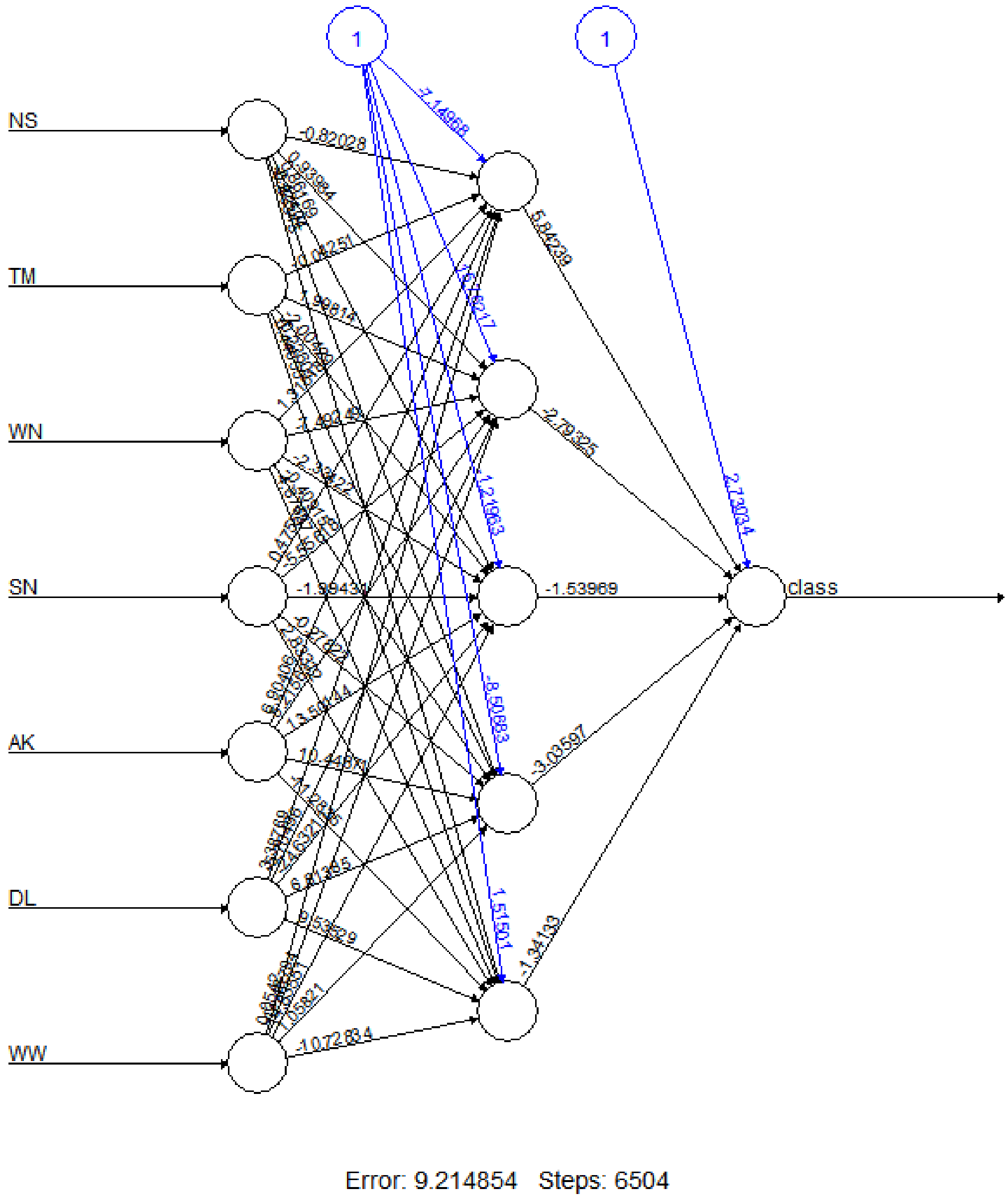
```
nn_5 <- neuralnet(myformula, train_recoded, hidden=c(5), lifesign="full", threshold=0.01,
                  linear.output = F)
```

```
## hidden: 5    thresh: 0.01    rep: 1/1    steps:     1000  min thresh: 0.131891460608357
##                                                     2000  min thresh: 0.0631148541798546
##                                                     3000  min thresh: 0.0535348295558427
##                                                     4000  min thresh: 0.032243623280333
##                                                     5000  min thresh: 0.0211918156654165
##                                                     6000  min thresh: 0.0121266642453183
##                                                     6504  error: 9.21485  time: 21.02 secs
```

The ANN converged, and much faster now!

Let's explore the output, now

```
plot(nn_5, rep = "best")
```



Nice!
Notice how each input flows into the network, is contributed to the hidden layer based on some weights (the numbers on the lines), and how the results at each node of the hidden layer are finally recomposed in the output layer (1 neuron) and returned as output. The blue circle and lines represent the *bias*.

YOUR TURN - Search the internet, and see if you can quickly figure out WHY we need a bias.

3.3 Interpret the output

Let's take a closer look at the weights:

```
nn_5$weights
```

```
## [[1]]
## [[1]][[1]]
##              [,1]        [,2]        [,3]       [,4]        [,5]
```

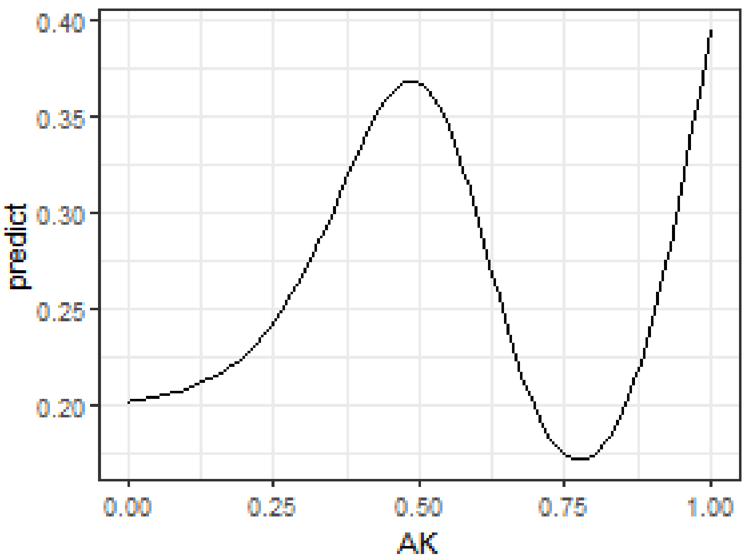
```
## [1,] -7.14968199  15.7621659  -1.2196303 -8.5068251   1.5150070
## [2,] -0.82028113   0.9398354   0.8616862 -2.4240391  -0.8334524
## [3,] -0.04251358   1.9981358  -2.0042855 -0.2260092   0.4405506
## [4,]  1.31617509  -7.4924928  -2.3342242 -0.4091491   4.8763654
## [5,]  0.47554612  -5.5561834  -1.9943108 -0.2782223   2.8331241
## [6,]  6.90406295   5.2156336  13.5014366 10.4487060 -11.2834992
## [7,]  3.38769447  -5.7049584 -24.6321131  6.8138519   9.5352948
## [8,]  0.05420349 -13.4679364  14.8355104  1.0582137 -10.7283368
##
## [[1]][[2]]
##           [,1]
## [1,]  2.730335
## [2,]  5.842394
## [3,] -2.793245
## [4,] -1.539685
## [5,] -3.035968
## [6,] -1.341332
```

It's not so easy to understand how these weights are combined in practice to return the habitat suitability of the corn bunting.
ANN are famous for being *black boxes*. Indeed the interpretation of the weights is not so straightforward. Yet, we can see that some weights are larger than others. Let's focus, for instance, on the weights of the sixth input (don't consider the bias, i.e., the first row, for now), i.e., `AK` which is the share of cropland. Understanding the combined effect of this input across ALL pathways is pretty challenging!

ANN are great tools for *predicting* something, but are not the best way of testing hypotheses!
This justifies the saying *'NN are the second best way to solve any problems. The first one is to actually understand a problem'*

There's a turnaround, though, to get a glimpse of the effect of a variable on the overall likelihood that a specific pixel is suitable habitat. We can simply create a new dataset, where we set all variables at their respective means, except for the one variable of interest. For the latter, we create a sequence of values spanning through the whole original range (0-1 in our case).
As an example, let's see what happens when the share of cropland (=`AK`) in a pixel increases, *all else being equal*.

```r
# create a new dataset
newdata <- as.data.frame(t(apply(scaled, MARGIN=2, "mean")))
newdata <- newdata[rep(1,100),] #repeat the means 100 times
newdata$AK <- seq(0,1, length.out = 100)
# feed the new dataset to the NN, and predict the outputs
predict_newdata = neuralnet::compute(nn_5, newdata)
newdata$predict = predict_newdata$net.result
# plot
ggplot(data=newdata) +
  geom_line(aes(x=AK, y=predict)) +
  theme_bw()
```



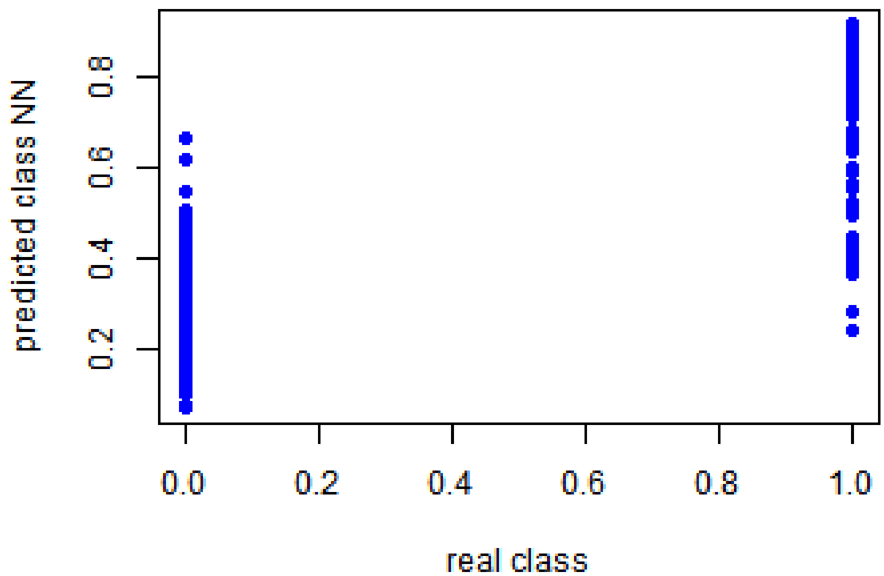
From the graph, we can see how the likelihood of a pixel to be classified as 'suitable' varies when the share of agricultural land increases.

YOUR TURN - How do your predictions vary for increasing `AK` in your model? Does it differ from what shown here? Why in your opinion? Try also recalculating the same curve, but for `WW` (i.e., the share of meadows in a grid cell) all else being equal.

3.4 Measure the NN's performance

We are not done, yet. We have to understand how our network performs. To do this, we have to calculate the error based on the `test` dataset. Here, we calculate **Sum of squared errors** (SSE) and the **Mean Square Error** (MSE), i.e., the average squared difference between our NN's predictions, and the known answers from the test data. We need to be consistent, though. Therefore, we first recode also the test dataset to 0.2 & 0.8.

```
#transform test data to 0.2 - 0.8
test_recoded <- test
test_recoded$class <- test_recoded$class*0.6+0.2

## Feed the test data into our ANN, and 'predict' the output
predict_testNN = neuralnet::compute(nn_5, test_recoded)
predict_testNN = predict_testNN$net.result

# Visual comparison of predicted vs expected values
plot(test$class, predict_testNN, col='blue', pch=16, ylab =
       "predicted class NN", xlab = "real class")
```




```
#same in tabular data
predict_01 <- ifelse(predict_testNN<0.5, 0, 1)
table(predict_01, test$class)
```

```
##
## predict_01   0   1
##          0 503  11
##          1   7 146
```

```
# Calculate Root Mean Square Error (RMSE)
SSE = sum((test_recoded$class - predict_testNN)^2)
RMSE = (SSE / nrow(test)) ^ 0.5
```

Not bad. We correctly classified almost all the entries of our test data. Our test data has a SSE = 6.0508887 and a RMSE = 0.095246.

YOUR TURN - Can you think of a possible reason why the `SSE` we obtained here differs so much from the Error reported in the graph above?

Congratulations! You've just trained your first ANN.

3.5 Test alternative configurations

How do we know whether ours is the best possible ANN?
There's no silver bullet, unfortunately (remember the Universality Theorem?). We have to rely on trial and error. But we need a way to reliably assess the performance of a specific configuration, so that we can make some comparisons. Remember, we only tested our ANN on one of the (almost) infinite number of subsets of our data. How does our ANN perform *on average*? This can be quantified using *cross-validation*.

Let's prepare a function to extract the RMSE, given a dataset, a formula and a configuration for our ANN, so that it will be easier to repeat this step multiple times later.

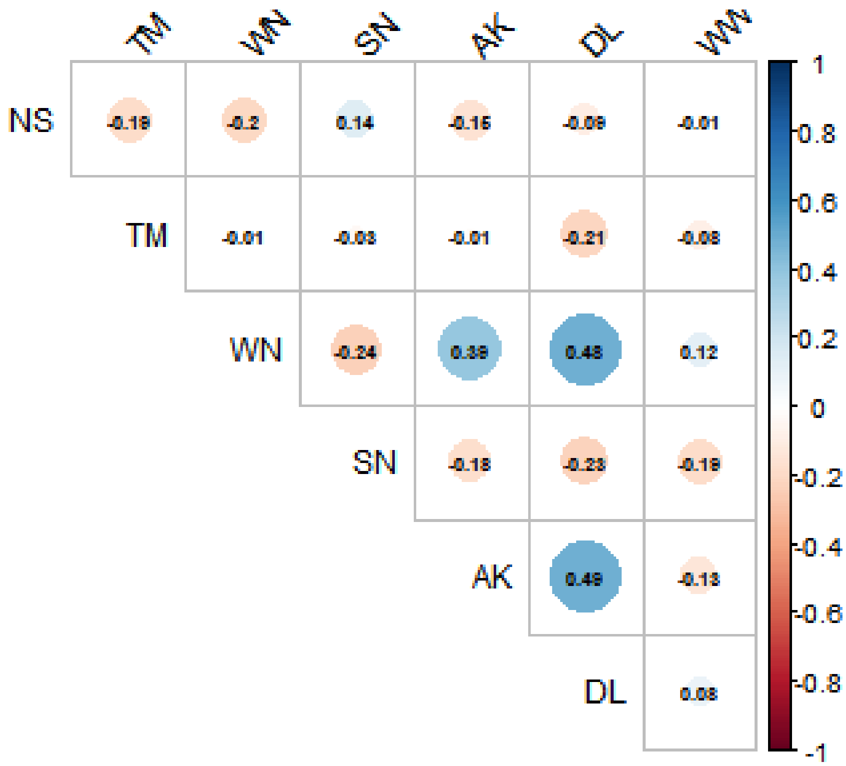
```
mydata.subset$class <- mydata.subset$class*0.6+0.2 #recode to improve performance

get.RMSE <- function(i, data, form, hidden, share.train=0.8, label="full"){
  train.id <- sample(1:nrow(data), n*share.train, replace=F)
  train = data[train.id,]
  test = data[-train.id,]
  nn <- neuralnet(form=form, train, hidden=c(hidden),
                  lifesign = "full", threshold=0.01, linear.output = F)
  predict_testNN <- neuralnet::compute(nn, test)
  predict_testNN = predict_testNN$net.result
  RMSE.NN = (sum((test$class - predict_testNN)^2) / nrow(test)) ^ 0.5
  #print(i)
  return(data.frame(config=paste(label, paste(hidden, collapse="_")), RMSE=RMSE.NN))
}
```

The formula above, takes a dataset, splits into a train and test subset, runs a NN, based on a given formula (`form`) and a given configuration, specified by the `hidden` vector. It finally returns the RMSE.
We can now run this formula, e.g., 5 (it'd be better to do it 10>) times, and compare the output of a NNs with different number of nodes. We also try and see what happens when we add an additional hidden layer.

```
#runs in about ~10-15 minutes
set.seed(200)
RMSE.2 <- lapply(1:5, get.RMSE, mydata.subset, myformula, c(2))
RMSE.3 <- lapply(1:5, get.RMSE, mydata.subset, myformula, c(3))
RMSE.4 <- lapply(1:5, get.RMSE, mydata.subset, myformula, c(4))
RMSE.5 <- lapply(1:5, get.RMSE, mydata.subset, myformula, c(5))
RMSE.6 <- lapply(1:5, get.RMSE, mydata.subset, myformula, c(6))

RMSE.2.2 <- lapply(1:5, get.RMSE, mydata.subset, myformula, c(2,2))
RMSE.3.3 <- lapply(1:5, get.RMSE, mydata.subset, myformula, c(3,3))
## compile and visualize output
RMSE.df <- do.call(rbind, c(RMSE.2, RMSE.3, RMSE.4, RMSE.5, RMSE.6, RMSE.2.2, RMSE.3.3))
```

Before looking at the output, let's also try out whether removing one variable improves the overall performance. Let's check first if there's any redundant (i.e., correlated) variables.

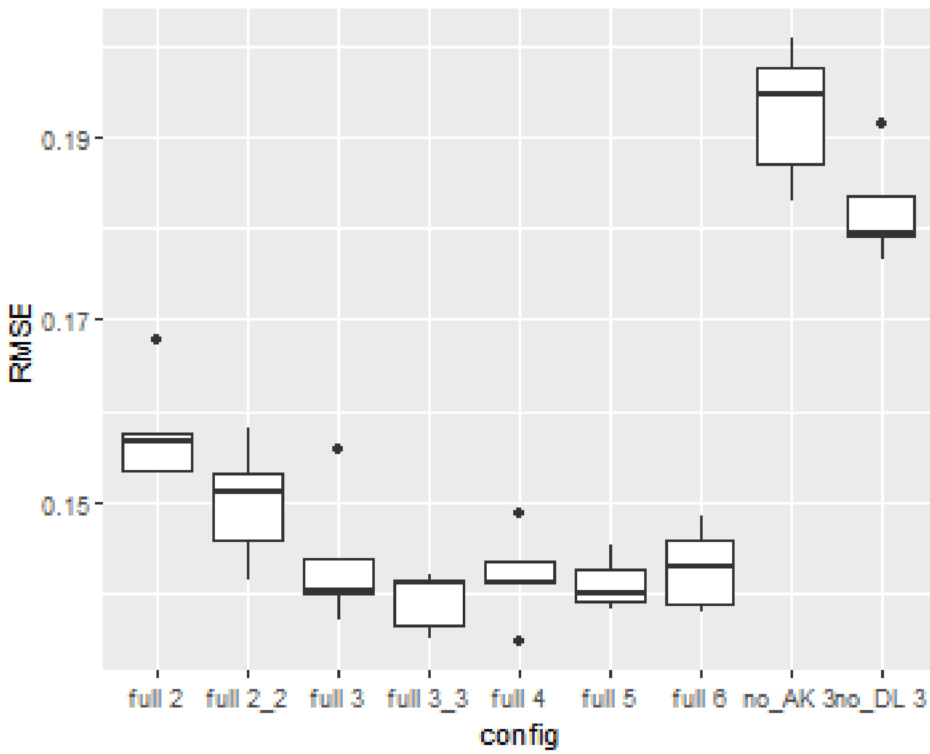
```
res <- cor(scaled, use = "pairwise.complete.obs")
corrplot(res, type = "upper",
         tl.col = "black", tl.srt = 45, number.cex=0.6, addCoef.col = "black", diag=F)
```



The area of loamy soils (`DL`) is negatively correlated both with the share of arable fields (`AK`) and with the distance to forest `WN`. We might wonder whether the performance improves when removing these variables.

```
RMSE.3.no_AK <- lapply(1:5, get.RMSE, mydata.subset,
                       as.formula("class ~ NS + TM + WN + SN + DL + WW"), c(3),
                       label="no_AK")
RMSE.3.no_DL <- lapply(1:5, get.RMSE, mydata.subset,
                       as.formula("class ~ NS + TM + WN + SN + AK + WW"), c(3),
                       label="no_DL")
RMSE.df2 <- rbind(RMSE.df, do.call(rbind, c(RMSE.3.no_AK, RMSE.3.no_DL)))
```

```
ggplot(data=RMSE.df2) +
  geom_boxplot(aes(x=config, y=RMSE))
```



When using too few nodes, the NN has a very bad performance (i.e., high RMSE). When using more than three nodes, the RMSE interval is largely overlapping across configurations, though. The NN with two layers of three nodes each seems to work slightly better than the others, although only marginally. Removing a variable, on the other hand, dramatically reduces performance.

YOUR TURN - Feel free to play around with the configuration, and number of variables used. Can you configure an ANN which works better?

3.6 Visual exploration of output

Let's now take a final look at our output, spatially. To do this, let's first make predictions on the whole study areas, using our original NN with 1 hidden layer, and 5 nodes.

```
predict_alldata = neuralnet::compute(nn_5, mydata.scaled)
predict_01 <- ifelse(predict_alldata$net.result<0.5, 0, 1)
predict_01 <- data.frame(PRED01=predict_01,
                         ZT_V100_=mydata$ZT_V100_ID)
# join output to fishnet grid
grid_sf2 <- left_join(grid_sf, predict_01, by="ZT_V100_")
```

We do the same for the reduced NN, without the variable `AK`.

```
myformula2 <- class ~ NS + TM + WN + SN + DL + WW
nn_3_noAK <- neuralnet(form=myformula2, train_recoded, hidden=c(3),
                       lifesign = "full", threshold=0.01, linear.output = F)
```

```
## hidden: 3    thresh: 0.01    rep: 1/1    steps:    1000  min thresh: 0.0662724252045939
##                                                   2000  min thresh: 0.0112793782386029
##                                                   2517  error: 36.16087 time: 6.03 secs
```

```
predict_alldata = neuralnet::compute(nn_3_noAK, mydata.scaled)
predict_01_noAK <- ifelse(predict_alldata$net.result<0.5, 0, 1)
```

```
predict_01_noAK <- data.frame(PRED01=predict_01_noAK,
                              ZT_V100_=mydata$ZT_V100_ID)

grid_sf3 <- left_join(grid_sf, predict_01_noAK, by="ZT_V100_")
```

Before plotting, let project also the continuous values of the habitat suitability index `HSI` onto our fishnet

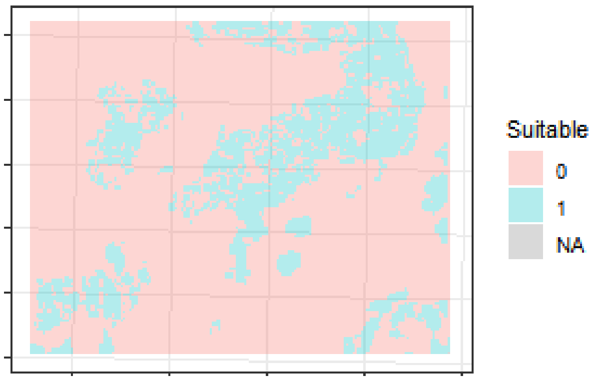
```
HSI <- mydata[,c(1,9)]
names(HSI)[1] <- "ZT_V100_"
grid_sf_hsi <- left_join(grid_sf, HSI,  by="ZT_V100_")
```

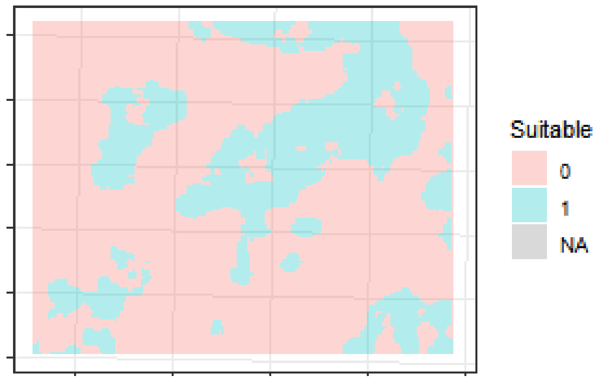
Prepare graphs

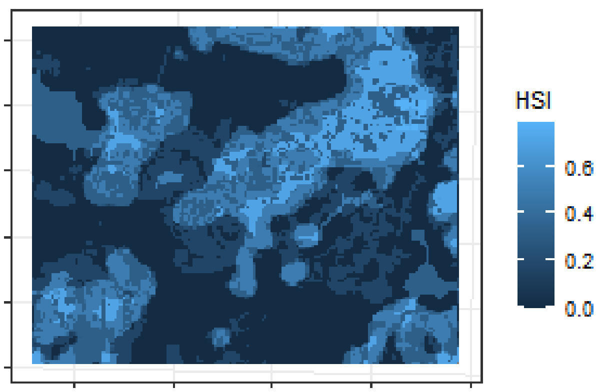
```
pred.full <- ggplot(data=grid_sf2) +
  geom_sf(aes(fill=as.factor(PRED01)), alpha=0.3, col=NA) +
  labs(fill='Suitable') +
  theme_bw() +
  theme(axis.text = element_blank())

pred.noAK <- pred.full %+% grid_sf3 #update graph with new data

HSI <- ggplot(data=grid_sf_hsi) +
  geom_sf(aes(fill=HSI), col=NA) +
  theme_bw() +
  theme(axis.text = element_blank())
```

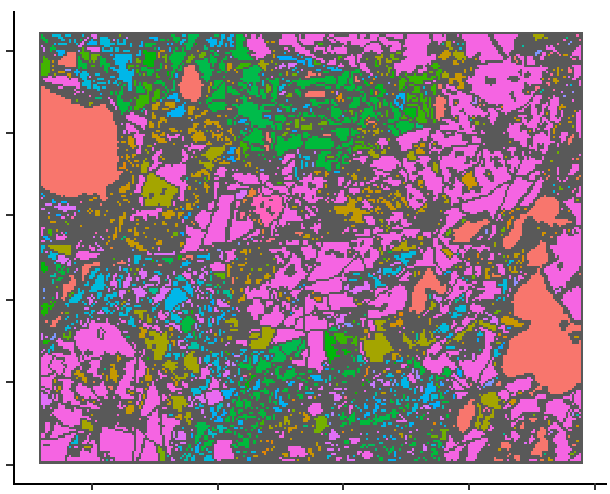
Create a panel to compare estimations to the map of land use.

```
cowplot::plot_grid(pred.full, pred.noAK,
                   HSI, Landuse + theme(axis.text = element_blank()),
                   nrow=2, rel_heights = c(1,1,1,0.6))
```





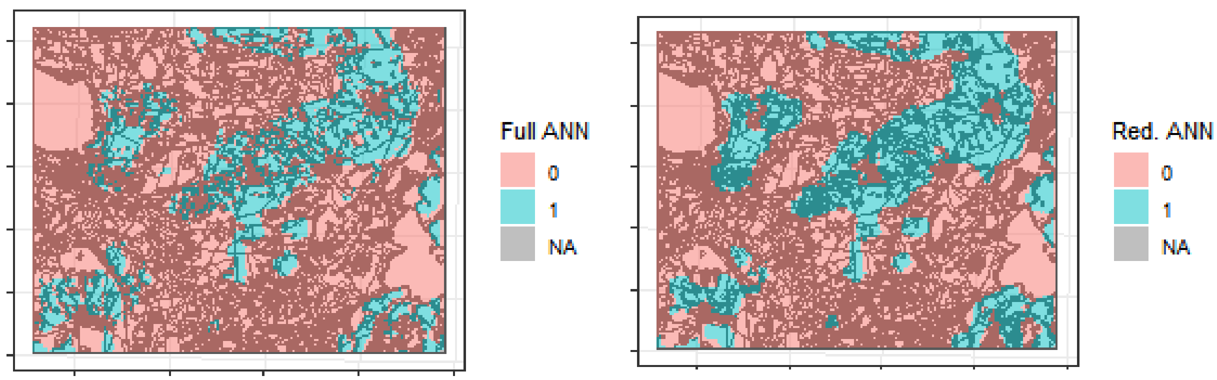




Overlap our prediction with the land use map

```
left <- ggplot(data=Biotopes_sf) +
  geom_sf(fill=NA) +
  geom_sf(data=grid_sf2, aes(fill=as.factor(PRED01)), alpha=0.5, col=NA) +
  labs(fill='Full ANN') +
  theme_bw() +
  theme(axis.text = element_blank())
right <- ggplot(data=Biotopes_sf) +
  geom_sf(fill=NA) +
  geom_sf(data=grid_sf3, aes(fill=as.factor(PRED01)), alpha=0.5, col=NA) +
  labs(fill='Red. ANN') +
  theme_bw() +
  theme(axis.text = element_blank())
```

```r
cowplot::plot_grid(left, right, nrow=1)
```



YOUR TURN - How does the output compare across the two ANN configurations? How do they relate to the map of HSI? In what habitat types is the Corn Bunting predicted to occur with higher frequency?

3.7 Export output

Let's export our output. We can both export it as a simple .csv, and converting it to a shapefile, to be used in any GIS softwares

```r
dir.create("_output")
st_write(grid_sf2, dsn="_output/CornBunting_NN3_pred.shp", append=T)
st_write(grid_sf3, dsn="_output/CornBunting_NN3_noAK_pred.shp", append=T)

write.csv(predict_01, file = "_output/CornBunting_NN3_pred01.csv")
write.csv(predict_01_noAK, file = "_output/CornBunting_NN3_noAK_pred01.csv")
```

. . . .and we're done!. . . . Thanks everybody.

4 Resources

- Daniel Shiffman - Youtube Playlist: The coding train - https://www.youtube.com/playlist?list=PLRqwX-V7Uu6aCibgK1PTWWu9by6XFdCfh
- Daniel Shiffman. The Nature of code (Chapter 10) - 2012 https://natureofcode.com/book/chapter-10-neural-networks/
- Machine Learning for Artists - https://ml4a.github.io/ml4a/how_neural_networks_are_trained/
- Miroslav Kubat. An introduction to Machine Learning - Springer 2017 https://www.springer.com/gp/book/9783319348865

5 SessionInfo()

```r
sessionInfo()
```

```
## R version 4.0.1 (2020-06-06)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18362)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.1252
## [2] LC_CTYPE=English_United Kingdom.1252
## [3] LC_MONETARY=English_United Kingdom.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] neuralnet_1.44.2 cowplot_1.0.0    ggplot2_3.3.2    sf_0.9-4
## [5] readxl_1.3.1     corrplot_0.84    dplyr_1.0.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.4.6       cellranger_1.1.0   pillar_1.4.4       compiler_4.0.1
##  [5] class_7.3-17       tools_4.0.1        digest_0.6.25      evaluate_0.14
##  [9] lifecycle_0.2.0    tibble_3.0.1       gtable_0.3.0       pkgconfig_2.0.3
## [13] rlang_0.4.6        cli_2.0.2          DBI_1.1.0          yaml_2.2.1
## [17] xfun_0.15          e1071_1.7-3        withr_2.2.0        stringr_1.4.0
## [21] knitr_1.29         generics_0.0.2     vctrs_0.3.1        classInt_0.4-3
## [25] grid_4.0.1         tidyselect_1.1.0   glue_1.4.1         R6_2.4.1
## [29] fansi_0.4.1        rmarkdown_2.3      farver_2.0.3       purrr_0.3.4
## [33] magrittr_1.5       scales_1.1.1       ellipsis_0.3.1     htmltools_0.5.0
## [37] units_0.6-7        assertthat_0.2.1   colorspace_1.4-1   labeling_0.3
## [41] utf8_1.1.4         KernSmooth_2.23-17 stringi_1.4.6      munsell_0.5.0
## [45] crayon_1.3.4
```