Report on Deep Learning analyses on a weather dataset

by Giuseppe Insana, February 2022

The dataset

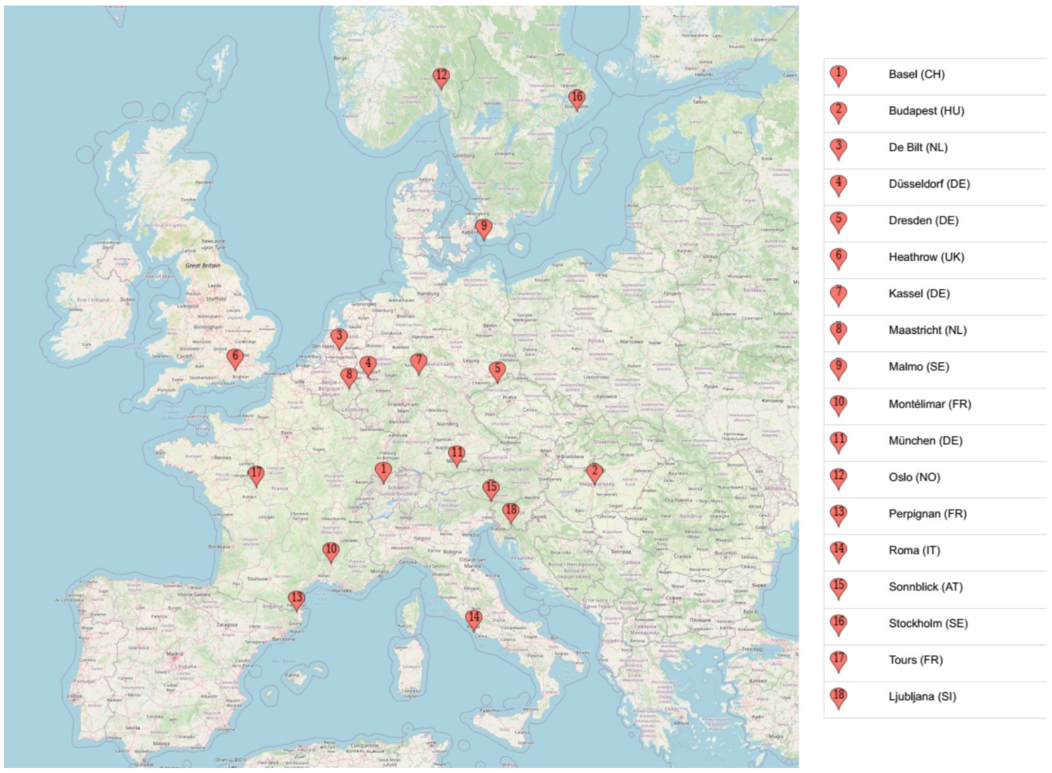
I decided to work on a weather dataset, chiefly because I never worked on this kind of data before (having mostly dealt with biological and linguistic analyses in the past).

The weather dataset analysed in this report has been created by Huber Florian from ECA&D data (see section on Data origin at the end for references).

It contains daily weather observations from 18 different European weather stations through the years 2000 to 2010.

The description of the data set says that the minimal set of variables 'mean temperature', 'max temperature' and 'min temperature' are available for all locations. An additional number of measured variables ('cloud_cover', 'wind_speed', 'wind_gust', 'humidity', 'pressure', 'global_radiation', 'precipitation', 'sunshine') are provided, but not for all the locations.

The following map shows the locations which are included in the dataset:



Objective of the analysis

We will try to predict tomorrow's weather for one location based on today's weather measures across all locations. In this analysis we will focus chiefly on trying to achieve prediction accuracy with deep learning models, as we have already gained several insights about the major factors that guide the predictions in previous analyses (presented in the other reports of this specialisation).

In particular we will try to predict whether tomorrow is going to be sunny in the city of Basel, with how many hours of sunshine.

Data exploration

A comprehensive exploratory data analysis has been previously conducted on the model and presented in another report (please check the separate document, file named **EDAcourseproject_report_Giuseppe_Insana.pdf**). We will here briefly summarise the findings of that report to give a description of the data we will process and to show the rationale behind the actions we will take.

We have analysed the types and amount of the available data, checking ranges and distributions of all observations.

Data attributes

The original data is loaded into a pandas dataframe which has 3654 rows (one per each day) and 165 columns:

- DATE, with integer values from 20000101 to 20100101, corresponding to interval from Jan 1st 2000 to Jan 1st 2010
- MONTH, integer 1 to 12
- and another 163 columns for the weather measurements at the different locations.

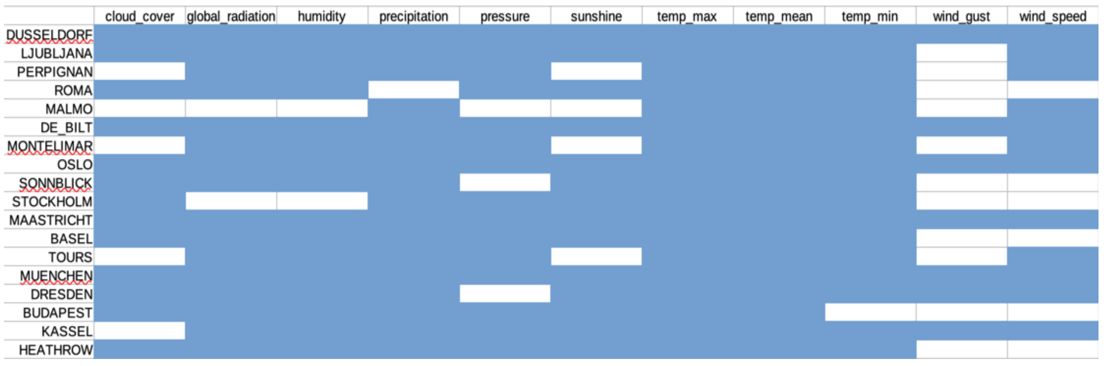
The measurements are labelled as LOCATION_*measure* (e.g. BASEL_cloud_cover, BASEL_pressure, OSLO_precipitation...) with the measured variables being: **cloud_cover, global_radiation, humidity, precipitation, pressure, sunshine, temp_max, temp_mean, temp_min, wind_gust, wind_speed**

All the measurements are loaded as floating point numbers, with the exception of cloud_cover, loaded as integer.

The **physical units** for the variables are described as follows:

cloud_cover in oktas; **wind_speed** and **wind_gust** in m/s; **humidity** in fraction of 100%; **pressure** in 1000 hPa, **global_radiation** in 100 W/m^2; **precipitation** in 10 mm; **sunshine** in 1 Hours; **mean max** and **min temperature** in Celsius degrees.

The following table shows which measures are available for which location:



	cloud_cover	global_radiation	humidity	precipitation	pressure	sunshine	temp_max	temp_mean	temp_min	wind_gust	wind_speed
DUSSELDORF	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
LJUBLJANA	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		🟦
PERPIGNAN		🟦	🟦	🟦	🟦		🟦	🟦	🟦		🟦
ROMA	🟦	🟦	🟦		🟦	🟦	🟦	🟦	🟦		
MALMO				🟦			🟦	🟦	🟦		🟦
DE_BILT	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
MONTELIMAR		🟦	🟦	🟦	🟦		🟦	🟦	🟦		🟦
OSLO	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
SONNBLICK	🟦	🟦	🟦	🟦		🟦	🟦	🟦	🟦		
STOCKHOLM	🟦			🟦	🟦	🟦	🟦	🟦	🟦		
MAASTRICHT	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
BASEL	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		
TOURS		🟦	🟦	🟦	🟦		🟦	🟦	🟦		🟦
MUENCHEN	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
DRESDEN	🟦	🟦	🟦	🟦		🟦	🟦	🟦	🟦	🟦	🟦
BUDAPEST	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦			
KASSEL		🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
HEATHROW	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		

This could obviously present problems of **unbalanced data**, in particular when extending the model to the prediction for different cities.

Null and Out-Of-Range values

The dataset does not contain missing values per se but there are several out-of-range values that can be considered as Null/Missing/Invalid observations:

cloud_cover measures should vary from 0 (sky completely clear) to 8 (sky completely clouded) oktas. But the data contains two data points (both for Stockholm, 20080724 and 20090625) with a value of -99 and one (again Stockholm, 20031108) with a value of 9

pressure: the data contains three entries (again for Stockholm, 20071008, 20000124, 20070603) with value of -0.099 and one entry (Tours, 20081230) with value of 0.0003. These out of range values can again be considered invalid and a decision should be taken for them akin to those mentioned before for cloud_cover.

sunshine: the data contains 29 negative values for hours of sunshine (again for the Stockholm location) which should be treated as invalid/null and dealt appropriately (as mentioned for cloud_cover and pressure). For the location of Oslo there are 24 measurements with more than 18 hours of sunshine, with 20 of them being 24h. While the northern latitude make very long daylight possible, this is for almost 18 hours in midsummer, while these huge values are from Nov-Dec 2006. We must hence treat these as wrong invalid data as well.

Distribution of values

After imputing the out of range values (shown below in the section on Data cleaning and feature engineering), the range, mean and standard deviation for all measures across all locations are the following:

measure	**mean**	**std**	**range**	
cloud_cover:	5.14	2.33	0.00 .. 8.00	*(okta)*
global_radiation:	1.37	0.95	0.01 .. 4.42	*(i.e. 1 to 442 W/m2)*
humidity:	0.75	0.14	0.10 .. 1.00	*(i.e. 1% to 100%)*
precipitation:	0.23	0.58	0.00 .. 16.04	*(i.e. 0 to 160.4 mm)*
pressure:	1.02	0.01	0.96 .. 1.05	*(i.e. 959 to 1016 hPa)*
sunshine:	5	4.41	0.00 .. 17.80	*(hours)*
temp_max:	14.5	9.58	-24.70 .. 41.10	*(°C)*
temp_mean:	10.39	8.41	-26.60 .. 33.10	*(°C)*
temp_min:	6.33	7.58	-30.30 .. 26.30	*(°C)*
wind_gust:	10.06	3.88	1.50 .. 41.00	*(m/s)*
wind_speed:	3.33	1.89	0.00 .. 16.30	*(m/s)*

The dataset was thoroughly explored visually by way of plots, to see the actual distribution of values and to gather insights, plotting the measured features as a whole or grouped spatially or temporally.

Major findings:

- strong dependence of weather measures by month and by location (obviously) for both ranges, mean and variance; for example pressure has smaller variance in summer months compared to winter months;
- the seasonal component appears to be responsible for multimodality in certain features;

- there are also year to year variations, with for example some years being on average warmer or colder; no overall trend was observed but this is probably due to the scale of the dataset (only ten years period);
- precipitation and wind_speed are the most skewed measures (more than 0.75 skew value) across all locations and for some locations humidity and cloud_cover as well. The most strongly skewed features are:

DRESDEN_precipitation	13.077
PERPIGNAN_precipitation	10.781
MONTELIMAR_precipitation	7.479
BUDAPEST_precipitation	5.662
MALMO_precipitation	5.337
MUENCHEN_precipitation	5.206
BASEL_precipitation	4.529
STOCKHOLM_precipitation	4.481
TOURS_precipitation	4.233
LJUBLJANA_precipitation	3.828

Correlations

The correlation between features was analysed, both for measures in a single location or across different locations.

The major insights were:

- related variables have (as expected) very high correlation:
  - temp_mean with temp_min and temp_max
  - wind speed and wind gust
- global radiation has high positive correlation with sunshine
- global radiation and sunshine correlate negatively with humidity
- precipitation appears to have extreme values concentrated where pressure has average values
- locations nearby have measures more highly correlated compared to locations geographically distant

For a complete analysis of correlated measures and how this was used to reconstruct the topology of the weather locations, please check the separate document, file named **unsupervised_courseproject_report_Giuseppe_Insana.pdf**).

Data cleaning and feature engineering

Out of range values

To clean the dataset we first dealt with the **out-of-range values** identified for the cloud_cover, pressure and sunshine measures.

The number of these invalid values appear in 1.64% of the total rows but as they only affect one location at a time they constitute only 0.01% of the total values. In the previous report we **recommended to not drop** the whole days' measurements but **instead to impute** the invalid values.

To do so we wrote code which gathers the values for the involved measure and location on the 5 days before and 5 days after the date of the out-of-range value. These values are then averaged and the average is used to impute the invalid value. When the invalid values appeared in succession the strategy is modified to use the window of 10 days (5 before and 5 after) but for the year before.

For example, the code identifies the above mentioned out-of-range values for pressure:

idx	DATE	STOCKHOLM_pressure	TOURS_pressure
23	20000124	-0.0990	1.0234
2710	20070603	-0.0990	1.0205
2837	20071008	-0.0990	1.0242
3286	20081230	1.0328	0.0003

and proceeded to impute them as following:

```
 ** Imputing 3 value(s) for column STOCKHOLM_pressure
 idx23: from -0.099 to 1.00449 using mean data from range 18-29
 idx2710: from -0.099 to 1.02079 using mean data from range 2705-2716
 idx2837: from -0.099 to 1.0203099999999998 using mean data from range 2832-2843

** Imputing 1 value(s) for column TOURS_pressure
idx3286: from 0.0003 to 1.02485 using mean data from range 3281-3292
```

Outliers

The **main outliers** identified in the EDA report were:

- the very low values for humidity, with majority of outliers from Sonnblick and (in much lower proportion) Perpignan
- the precipitation extremes
- the highest recorded wind_speed measures (which are almost all from Perpignan location)

These could represent extreme weather conditions, which may or may not hinder the prediction abilities of the ML models.

Similarly to what said for out-of-range values, the outliers account for a negligible portion of the dataset but non-negligible number of rows. Thus it would be best to impute the single values (using the same strategy as above, averaging over a window of days) rather than excluding the whole rows.

As previously observed during regression and classification work on this dataset, imputing these outliers made negligible difference to the results of the trained models.

Feature engineering

The original **dataset does not contain categorical data** which would need to be converted (e.g. by one-hot encoding).

The DATE information is dropped from the dataset, leaving 164 columns (MONTH and weather measures across locations).

In the pipelines we will use, we will be adding **transformations** (transforming the most skewed variables outlined above) and **scaling** of the features.

Models training

Procedure

At the beginning of the analysis we set aside a random sample of 20% of the entire dataset to be used as **test** for evaluation. The remaining 80% is the data on which we **train** each model on.

The models were created using ***keras*** functional API. The input layer was set to accept a vector of 164 features (or a reduced number of features for the alternative strategies discussed later on) and the output layer was a single dense layer of size 1. Different model architectures were examined by training, comparing and presenting the results. Unless differently specified, the activation function for the layers was set as '*relu*' and no activation was added to the last output layer.

Each model was trained for a maximum of 200 epochs, but often early stopping would terminate training much earlier, as will be explained below.

Different batch sizes were tried, finally settling for 256. Seeding the random generator (as explained in the next paragraph) made this choice less relevant. '*adam*' was used as the optimizer and '*mae*' (mean absolute error) as loss function. The metrics of *mse* and *mae* were recorded for train and test sets during fitting.

In order to be able to correctly compare the different models trained and for having reproducible results, the random generators were re-seeded with the **same seed** before each model got trained. This ensured that the data was presented in the same order for each model (although shuffled at each epoch, as the order is determined by the random generator).

At the same time, to lower the possibility of pure chance dominating the evaluation of the performance, all models were trained **five** times, varying the global seed at each iteration, and then averaging the scores among the five runs. Thus all scores presented are the average of five different training of the model.

For **scoring**, two metrics have been used: the mean squared error (MSE) and the R^2 score (coefficient of determination).

With $\hat{y}_i$ being the predicted value of the ith sample, and y_i as the corresponding true value, then the mean squared error (MSE) estimated over n samples is defined as:

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_\text{samples}} \sum_{i=0}^{n_\text{samples}-1} (y_i - \hat{y}_i)^2$$

while the R^2 score is defined as:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance.

The train data metrics MSE and R^2 (indicated respectively as columns **trainMSE** and **trainR2** in the tables below) are computed over predictions from train data, while the test data scores (indicated as **testMSE** and **testR2**) are computed against the test data which has been set aside.

In several cases we present **plots of predicted vs true measure** for both train and test sets which often highlight overfitting problems.

Naming convention

The model names, appearing in the tables below and as title of the plots, reflect the hidden layers used in the model architecture (e.g. "100-50-10" indicating three dense layers of 100, 50 and 10 units respectively) and also the transformations operated on the input data. For example models whose name start with "ctss" reflect that the input data has the most skewed columns transformed with *log1p* and scaled with *StandardScaler*, while those whose name starts with "ss" were trained with scaled input but without correcting features with skewed distributions.

Scaling

We tested both scaling the features with *StandardScaler* and *MinMaxScaler*. All models trained with features scaled with *MinMaxScaler* performed consistently worse, with R^2 score on test data between 0.03 and 0.15 worse than the corresponding models where StandardScaler was used. Thus *StandardScaler* was preferred. We also tried passing un-scaled data, but performance was very bad, as expected due to the very diverse nature of the weather measures.

We also tried the addition of batch normalization initial layer. This is indicated in models containing "bn" in the name. As expected, since the data was already scaled, the addition of batch normalization didn't make much of a difference, with on average a 0.001 higher R^2 score on test data for all models.

Column Transformations

We tested adding the transformer *log1p* (using sklearn's *ColumnTransformer*) to the most skewed features. We tried two different setups: transforming those with skew value more than 0.75 (41 features) or those with skew value more than 0.5 (70 features).

The performance of ctss models compared to the ss models (without column transformation) was consistently better, with on average 0.006 higher R^2 score on test data and a maximum obvserved improvement of 0.03. On the other hand, the comparison between using 0.5 or 0.75 as skew value did not offer similar consistent results. Some of the models would perform slightly better when trained on the data where the 41 features whose distribution had skew value higher than 0.75 had been transformed, while others would perform better using the 0.5 skew_value (transforming 70 features). Still there was an average improvement of 0.004 R^2 score on test data using 0.75 but ranging from up to 0.06 better to 0.02 worse.

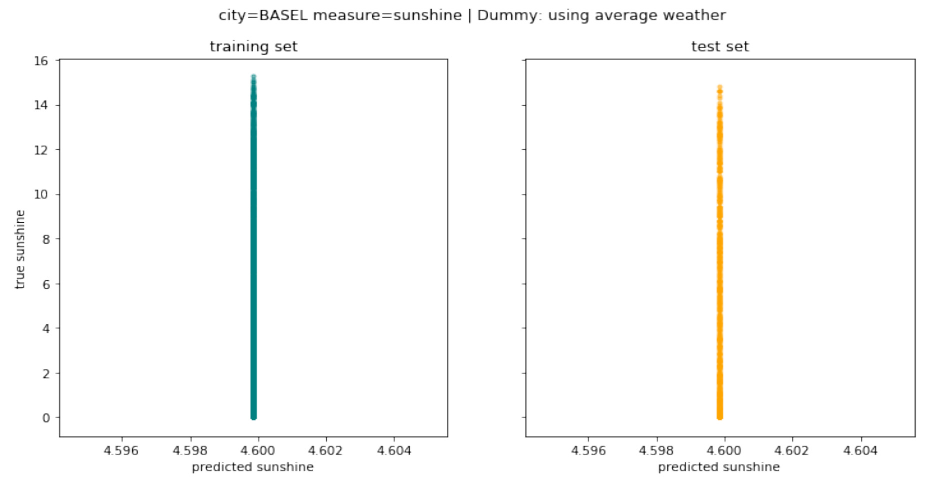
The 0.75 value was in the end adopted but, as just mentioned, the performance contributions were not consistent.

Baseline

To compare machine learning models it is often useful to have a baseline. The simplest approach is to use a dummy regressor, which always uses the mean for the measure as the prediction. For BASEL_sunshine the dummy regressor always predict 4.6 hours of sunshine, regardless.

It obviously performs extremely bad:

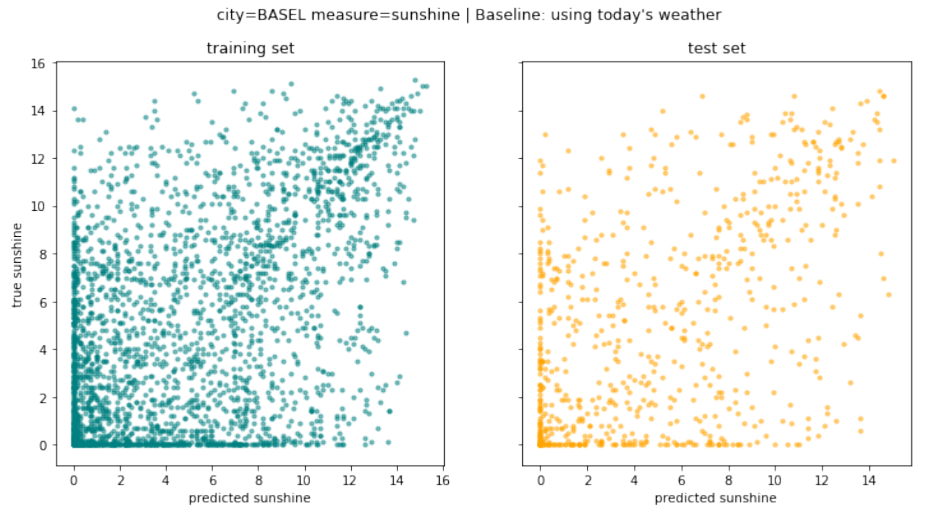
	trainMSE	trainR2	testMSE	testR2
dummy_mean	19.672	-0.004	19.597	-0.005



A better baseline we can devise is to use today's measured observation in order to predict tomorrow's weather. For example, if today the city's sunshine was measured in 5 hours, this baseline model would predict tomorrow's sunshine to also be 5 hours.

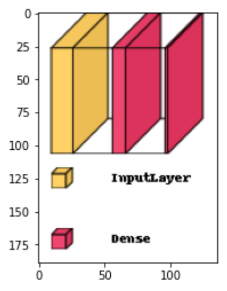
It performs better than the dummy regressor, but still poorly, showing that today's amount of sunshine is not a good indicator of the amount of sunshine we'll see tomorrow:

	trainMSE	trainR2	testMSE	testR2
same_as_today	17.494	0.056	18.917	0.030

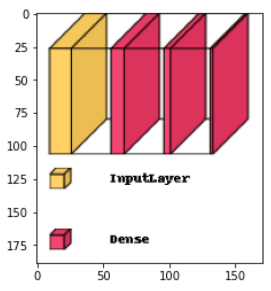


Simple models

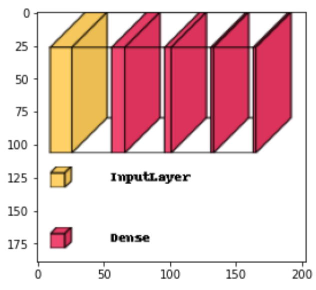
To begin with, we trained models with a single hidden layer of 100 units, then adding another layer with 50 units and then adding one more hidden layer with 10 units:



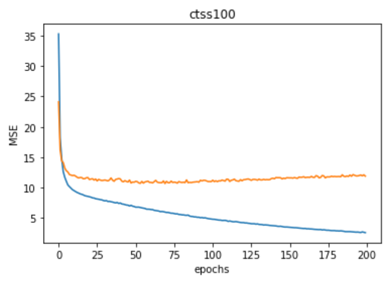
100

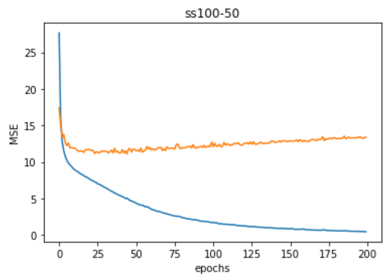


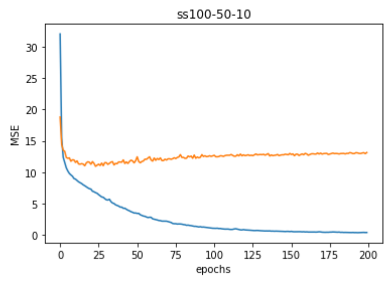
100-50



100-50-10





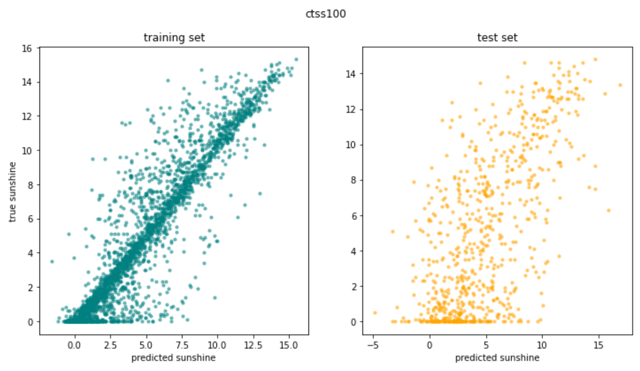


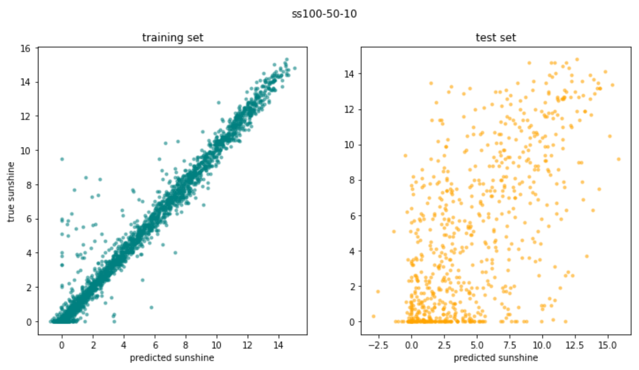
Plots of MSE vs Epoch during the training of the models. Blue: train data, Orange: test data (validation)

Adding hidden layers enables the model to fit better and better to the training data, with trainMSE decreasing down to 0.75 and trainR^2 reaching 0.96. This comes at the price of increasingly worse performance on test data (ranging from 0.32 to 0.39 R^2), showing clear overfitting:

	trainMSE	testMSE	trainR2	testR2
ss100	3.2496	12.0022	0.8247	0.3845
ctss100	3.3322	11.8581	0.8202	0.3919
ss100-50	1.1088	13.4727	0.9402	0.3091
ctss100-50	1.1168	13.1439	0.9397	0.3259
ss100-50-10	0.8236	12.8872	0.9556	0.3391
ctss100-50-10	0.7486	13.2680	0.9596	0.3196

This is also made quite evident by the plots of predicted vs real target variable, comparing the model with one hidden layer (left) to the one with three hidden layers (right):





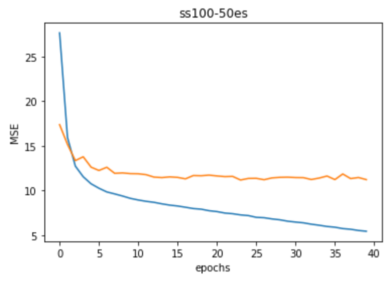
We will attempt different strategies to deal with overfitting and see whether we can improve the even the deeper models to increase the accuracy on test data.

Adding early stopping

As made clear by the plots of MSE score at each epoch of training, for deeper models the MSE on test data actually starts increasing after a while, while the MSE for training data continues to decrease. A possible strategy would then be to add early stopping: to interrupt training when validation scores have stopped improving.

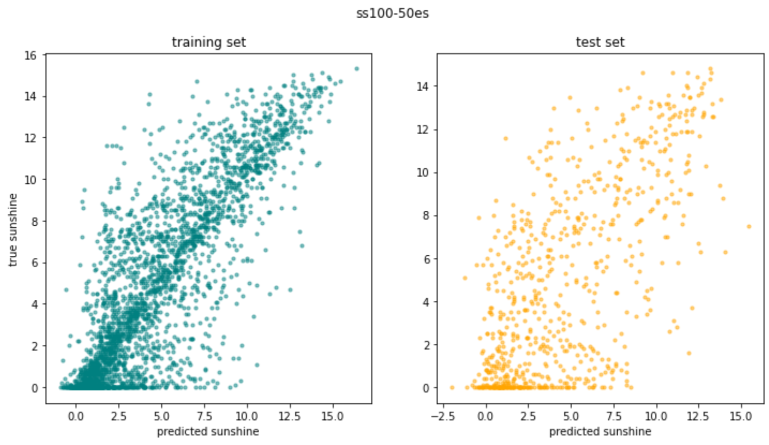
We added an *EarlyStopping* callback during training, monitoring validation loss with a patience of 15 (number of epochs with no improvement after which training will be stopped).

Early stopping interrupted training at about 60 epochs for the model with a single hidden layer, and at about 40 epochs for the other two models.



As shown by the table summarising the results, the training R^2 is brought down dramatically but the test R^2 improve to 0.40-0.44:

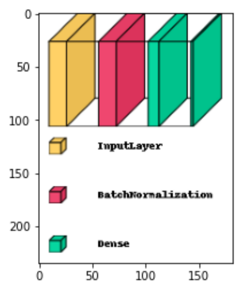
	trainMSE	testMSE	trainR2	testR2
ss100es	5.8662	11.0495	0.6835	0.4333
ctss100es	6.1049	10.9137	0.6706	0.4403
ss100-50es	5.0180	11.4254	0.7293	0.4141
ctss100-50es	4.6876	11.2256	0.7471	0.4243
ss100-50-10es	4.5565	11.6257	0.7542	0.4038
ctss100-50-10es	4.3769	11.5198	0.7639	0.4092



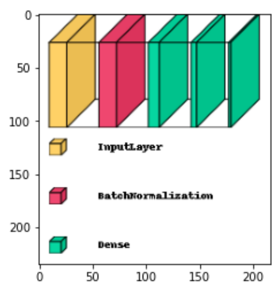
Overfitting is a little less pronounced..
but negative sunshine hours predicted??

Adding batch normalisation

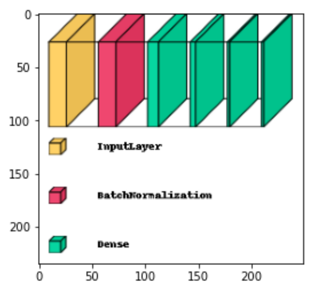
We also tried batch normalisation, which – as mentioned above – added a slight improvement, although not always consistently.



bn100



bn100-50



bn100-50-10

Still, for these models, it improved test R^2 scores by about 0.01:

	trainMSE	testMSE	trainR2	testR2
ctssbn100es	6.9722	10.6808	0.6238	0.4522
ctssbn100-50es	5.4877	11.1461	0.7039	0.4284
ctssbn100-50-10es	4.9478	11.2020	0.7331	0.4255

Adding L1 regularisation

To each layer a kernel weight regulariser was added, with the default L1 regularisation factor of 0.01

The train R^2 scores were brought down dramatically (about 0.52-0.53), and also the test R^2 scores showed in general slightly worse performance:

	trainMSE	testMSE	trainR2	testR2
ctss100l1-es	8.9726	11.0967	0.5159	0.4309
ctssbn100l1-es	8.9669	11.2080	0.5162	0.4252
ctss100l1-50l1-es	8.6778	11.4906	0.5318	0.4107
ctssbn100l1-50l1-es	8.8004	11.1305	0.5252	0.4292
ctss100l1-50l1-10l1-es	8.9726	11.0967	0.5159	0.4309
ctssbn100l1-50l1-10l1-es	8.7465	11.3955	0.5281	0.4156

As mentioned, it is not clear whether the batch normalisation helps or hinder the models' performance. In this case it didn't help.

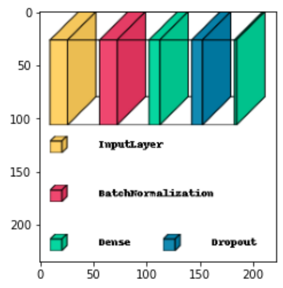
Interestingly, the performance of the deeper models start to be comparable to the shallow one.

Also we finally don't see anymore the impossible "negative sunshine hours" getting predicted by the models.

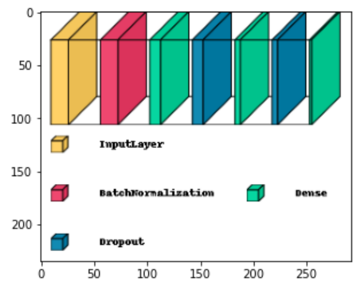
Adding dropout layers

Another possible strategy to avoid overfitting is to add dropout layers, which will randomly set input units to 0, with a specified frequency, at each step during training time.

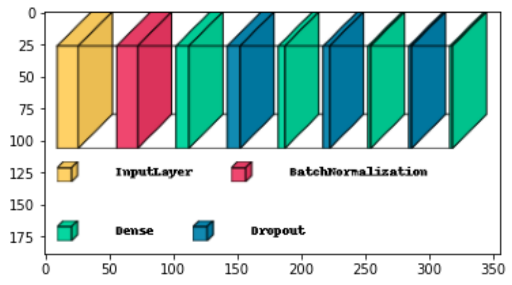
We tried adding dropout layers interleaved with the hidden layers



100l1-d40



100l1-d40-50l1-d10

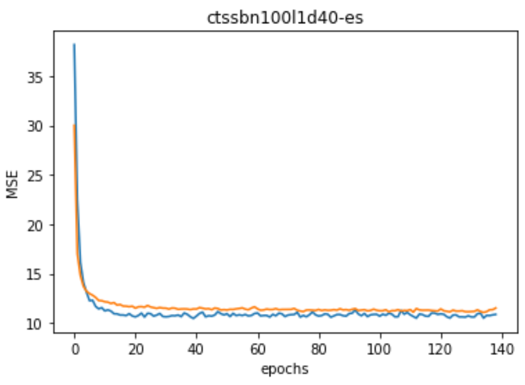


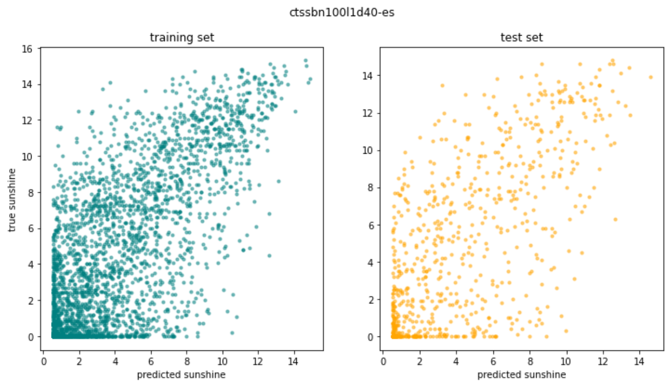
100l1-d40-50l1-d10-10l1-d10

Once more train R^2 scores decrease, while R^2 scores are almost the same, or slightly worse:

	trainMSE	testMSE	trainR2	testR2
ctssbn100l1d40-es	9.2859	11.1184	0.4990	0.4298
ctssbn100l1d40-50l1d10-es	9.3653	11.1625	0.4947	0.4275
ctssbn100l1d40-50l1d10-10l1d10-es	9.6789	11.4222	0.4778	0.4142

Overfitting seems to be kept at bay, as also shown by the following plots, but still there has not been much improvement in accuracy on the test data.

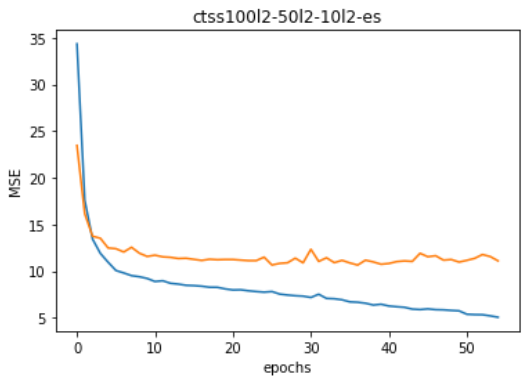


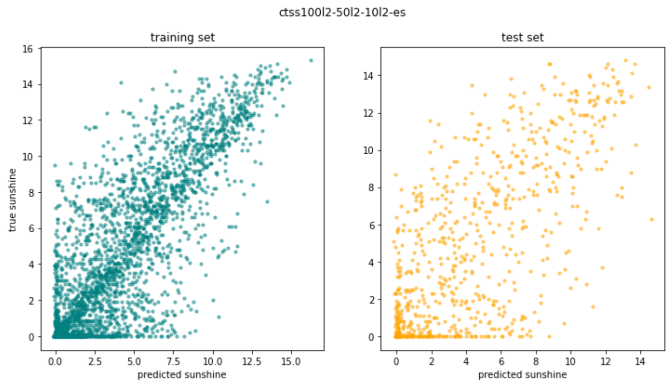


Using L2 regularisation

The use of L2 kernel regularisation offered an average improvement of 0.02 in test R^2 scores across the models when compared with L1. The train scores are also much higher. The value used for the L2 regularisation factor was the default one of 0.01.

	trainMSE	testMSE	trainR2	testR2
ctss100l2-es	6.8714	10.8280	0.6293	0.4447
ctss100l2-50l2-es	5.2706	10.9951	0.7156	0.4361
ctss100l2-50l2-10l2-es	6.0719	10.7372	0.6724	0.4494

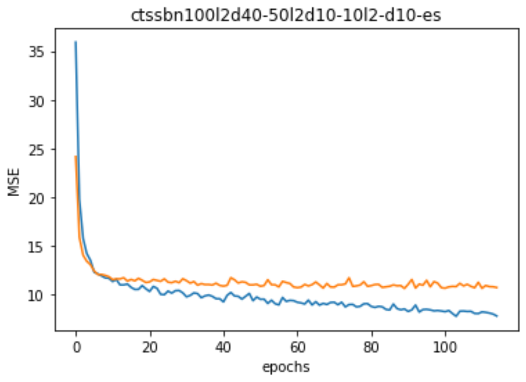


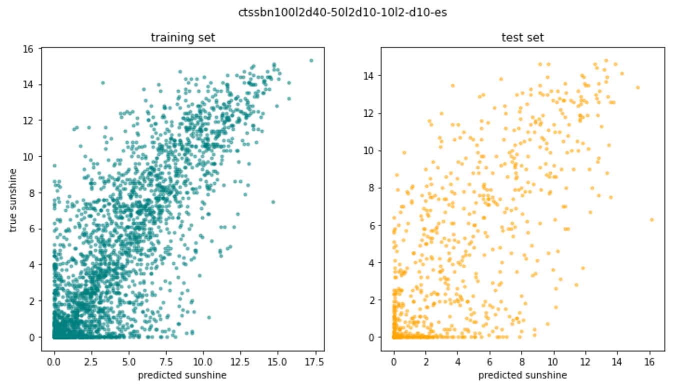


L2 regularisation and dropout layers

The combination of L2 regularisation and dropout adds another slight improvement to test R^2 scores and again confirms better performance of L2 compared to L1, for this data set. The train R^2 scores once more are lower, compared with the same models without dropout:

	trainMSE	testMSE	trainR2	testR2
ctss100l2d40-es	7.6295	10.8213	0.5884	0.4450
ctss100l2d40-50l2d10-es	6.2169	10.8200	0.6646	0.4451
ctss100l2d40-50l2d10-10l2-d10-es	6.6887	10.6874	0.6391	0.4519





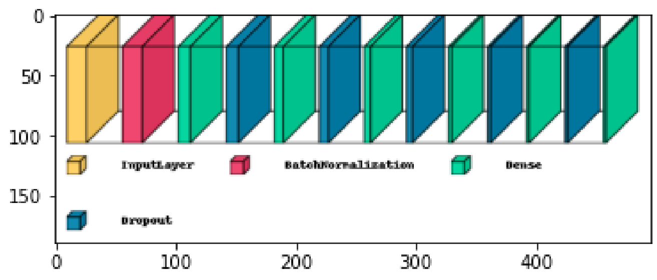
Using L1_L2 regularisation

We also tried using the combination of L1 and L2, using the *l1_l2* function that applies both penalties. Results were not better than those obtained using L2 alone, neither with nor without dropout layers. The test R^2 scores were in the range 0.41-0.43

Trying to go deeper

Adding more hidden layers to create a deeper network, but still using regularisation techniques to keep overfitting at bay, yielded overall lower performances, in the range of 0.40-0.43 test R^2.

For example:



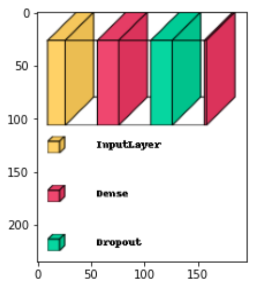
100l2-d40-75l2-d30-50l2-d20-25l2-d10-10l2-d10

	trainMSE	testMSE	trainR2	testR2
ctssbn100l2-d40-75l2-d30-50l2-d20-25l2-d10-10l2-d10es	6.3813	11.0446	0.6557	0.4336

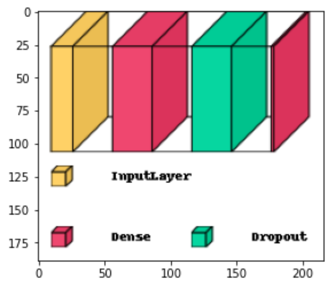
Going deep is not always the best strategy, especially when there is not a huge amount of training data.

Trying to go wider

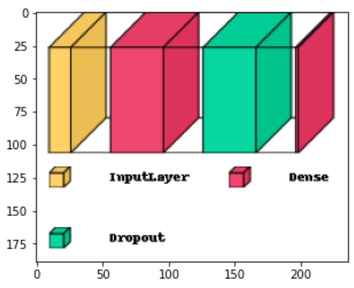
Another possibility, rather than adding hidden layers, would be to increase the number of units for each layer. This strategy turned out actually more promising:



200l2-d40



300l2-d40



400l2-d40

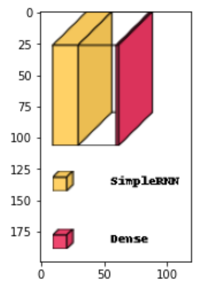
	trainMSE	testMSE	trainR2	testR2
ctss200l2d40es	7.2718	10.6178	0.6077	0.4555
ctss300l2d40es	6.7421	10.4508	0.6363	0.4640
ctss400l2d40es	6.6503	10.5822	0.6412	0.4573

Combining deeper with wider strategies did not offer further improvements.

RNN

We also tried to train recurrent neural networks. Although the data is not sequential, the fact that there is high correlation among the weather measures is a type of *context* in which the single weather measures are.

We tried both shallow networks with a single SimpleRNN layer of either 100 or 200 units and networks where the RNN was followed by dense layers. We used both *tanh* and *relu* as activation for the RNN layer but *tanh* activation performed worse than *relu*. The RNN models' R^2 scores were in the range 0.41-0.44 with the best being:



R200

	trainMSE	testMSE	trainR2	testR2
ctssR200es	7.2908	10.831	0.6067	0.4445

Alternative strategies

These are two alternative strategies we have tried and compared to what presented above:

1. Pre-selecting only a subset of features and locations (to create a balanced set), i.e. using only the measures 'global_radiation', 'humidity', 'precipitation', 'pressure', 'sunshine', 'temp_max', 'temp_mean' for the cities 'BASEL', 'BUDAPEST', 'DE_BILT', 'DUSSELDORF', 'HEATHROW', 'LJUBLJANA', 'MAASTRICHT', 'MUENCHEN', 'OSLO' for a total of 71 features;
2. Pre-selecting the features most correlated to the target feature, using a table of correlation.

For (1), using the balanced set resulted overall in a worse performance across all models tested, with the top models barely reaching 0.40 test R^2 scores. For example:

	trainMSE	testMSE	trainR2	testR2
ctssbn300l2d40es	8.9071	11.648	0.5195	0.4026

For (2), the columns with an absolute correlation value to the target feature of more than 0.5, sorted by absolute correlation are: 'TOURS_global_radiation', 'BASEL_global_radiation', 'MAASTRICHT_global_radiation', 'BASEL_sunshine', 'MONTELIMAR_global_radiation', 'TOURS_humidity', 'DUSSELDORF_global_radiation'
Using only those as input, and running all previously described models, resulted in even worse performance, with test R2 scores ranging between 0.33 and 0.35.

Final model recommendation

After testing extensively the models described above for all the strategies we outlined, we recommend using the ctss300l2d40es model (single hidden layer of 300 units, L2 regularisation with 0.01 factor, dropout layer with 40% rate, early stopping) trained over all the available columns scaled with standard **scaling** and transforming with **log1p** the most skewed features (>0.75 skew value).

As our objective was to aim for the most predictive power, that is the model with higher accuracy. It also is relatively fast to train, being quite shallow.

The test R^2 score for that model was 0.464. This is almost the same, only slightly worse than what obtained with *sklearn*'s ElasticNet which had achieved a test R^2 score of 0.470. That Regression model was trained over all the feature columns and their linear combinations using PolynomialFeatures of second degree for a total of 13860 input features (using hyper-parameters α=0.10 and l1_ratio=0.88).

Although the deep learning model came slightly lower in performance, it was much faster to train and only worked on the 164 original features.

Key findings and insights

We determined that although it is a very hard problem to predict tomorrow's weather, we can do much better with appropriate models compared to baseline assumptions.

It was also made very clear that neural networks are very prone to overfitting and adequate regularisation strategies need to be implemented to avoid this.

Scaling and transformation of skewed distributions are necessary to achieve better predictions.

Creating deep learning models is not a trivial task. In particular the number of possibilities for model architecture is so vast and this, coupled with the tuning of so many hyper-parameters, quickly creates a runaway abundance of models to be tested.

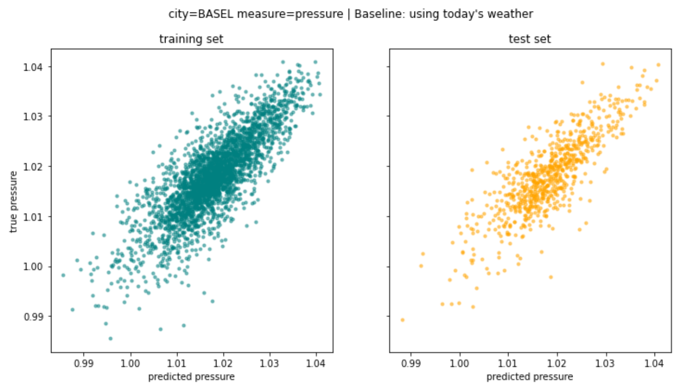
An interesting aspect to remark is that while all regression supervised ML models previously trained on this dataset made very bad predictions when the true value is 0 (where they instead predicted a certain number of sunshine hours), the neural network models (once L1 or L2 regularisation was added) are less biased in their errors, with the plots of true vs predicted values more symmetric compared to what previously observed during ML Regression analyses.

Next steps

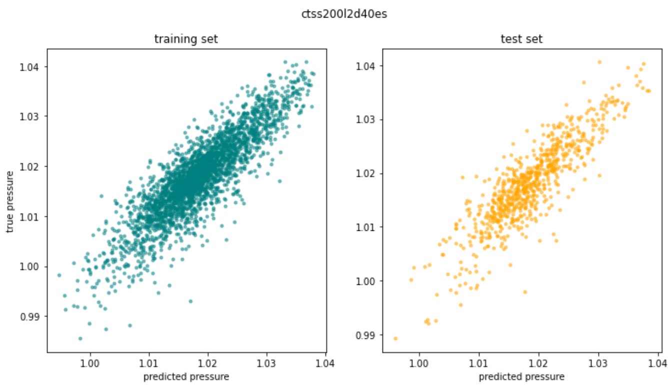
It would be both straightforward and interesting to retrain all models changing the target measure. In particular for two main reasons: firstly, to see which ones are easier to predict and which ones are harder; secondly, to see whether the fact that the dataset is originally unbalanced has a big impact in predicting weather measures for some cities.

Preliminary results show that for example pressure should be easier to predict, as it is more consistent from day to day.

The baseline "same as today" model in fact already achieves 0.65 test R^2: and one of the first tested models reaches 0.76:



baseline 'same_as_today'



200l2-d40

The most interesting extension would be to work with a time frame of observations to predict tomorrow's weather. Instead of using only today's measure across all locations, we could use a series of measures for subsequent days (e.g. the weather of the past *week*) to predict tomorrow's weather.

Finally, it is worth noting that deep learning models are known to benefit from a great abundance of training data. It is entirely possible that having access to more years of weather observations would improve the results.

Conclusions

The dataset looked like a very promising one to use for machine learning with limited amount of data cleaning necessary.

The major initial drawback was that only two features are available at all locations (temp_mean and temp_max), but it is too restrictive to only consider these two. To achieve a balanced dataset it would be needed to drop a combination of some features and some locations in order to maximise the quantity of data left. Nevertheless the models tested appeared to be quite able to cope with the unbalanced information and limiting to a balanced subset actually gave much worse results.

We also confirmed how training of neural networks must always be coupled with appropriate regularisation techniques to avoid heavily over-fitting models and showed the relevant contributions of different strategies.

Methods and Materials

Data manipulation and analysis was performed on a MacOS laptop using own written code in python language, working in a jupyter notebook and taking advantage of the following python libraries: pandas, seaborn, scikit-learn, tensorflow / keras, visualkeras

Data origin acknowledgement

Dataset compiled by Huber, Florian (2021); Zenodo. https://doi.org/10.5281/zenodo.4770937

ORIGINAL DATA TAKEN FROM:

EUROPEAN CLIMATE ASSESSMENT & DATASET (ECA&D), file created on 22-04-2021 THESE DATA CAN BE USED FREELY PROVIDED THAT THE FOLLOWING SOURCE IS ACKNOWLEDGED:

Klein Tank, A.M.G. and Coauthors, 2002. Daily dataset of 20th-century surface air temperature and precipitation series for the European Climate Assessment. Int. J. of Climatol., 22, 1441-1453. Data and metadata available at http://www.ecad.eu