# Machine learning: final project

I will write a summary of the contents of the book that I think are important for our project.

* Basics of Recurrent Neural Networks: Memory cells
* Input and output structure of the model
* Forecasting a Time Series and implementation of a Deep RNN
* Trend and Seasonality advice from the book
* Different methods to forecast several steps ahead.
* Book’s advice: MC Dropout for adding error to the data
* LSTM Cells
* GRU Cells

Other generalities:

* End-to-End machine learning project
* Dimensionality reduction
* Introduction to Artificial Neural Networks with Keras

## Basics of RNN: The memory cells

*Attention: Until the contrary is said, we will be talking of simple RNN, which means they only have one hidden layer. Later on we will start talking about Deep RNNs*

Since now, or at least until the initial project, we have only looked at Feed-forward Neural Networks, which means we give the network an input, this input has to pass through the next layer undergoing the corresponding operations and so on until an output is obtained. However, RNNs cells receive information both from the new input and from the output of the cell in the previous step (with step, we mean the previous time the network was fed with input data). Therefore, they are very useful for time series, as they take into account both the prediction of the previous time step and the input of the current time step to generate the output.

Basically: We have a set of weights “x” for the inputs and a set of weights “y” that are generated from the outputs.

*Equation 1:*

Where is the activation function, W are the weights (for the inputs and for the outputs of the previous time step) and b is a bias (like the independent term in a linear regression).

If we are training the model with an mini-batch of m instances (instances meaning the data of a specific time) then the variables have the following shapes:

* : 🡪 m because we will have a prediction for every instance from the input and n neurons because we are still talking about not deep neural networks so we just have one hidden layer
* 🡪 n inputs is often referred to as the input dimensionality because it refers to the number of variables each input carries
* 🡪 Those are the weights of the inputs to the neurons of the hidden layer
* 🡪 The output of every neurons has e weight over every neuron
* 🡪 vector containing the bias of every neuron

So, according to equation (1), depends on and , which at the same time depends on and and so on. We could say our neural network has some sort of **memory.** A part of a recurrent neural network that has this sort of memory is called a memory cell: an example of memory cell could be a simple recurrent neuron or a layer of recurrent neurons. These are very simple examples of memory cells, and they are capable of learning only very short patterns (typically about 10 steps long).

In RNNs it is very important to have in mind the concept of **state of the cell.** It is denoted by where h stands for “hidden” and it is a function of the state of the previous time step and the input at the current time step, . In the case of the basic cells we have discussed so far, the output of the cell is directly equal to the state. However, there are very complicated cells in which the output is not just the state but a complicated combination between the state and the input.

In LSTM cells for example, the cell has a long term memory term that decides which part of the input should be forgotten and which part of the input is important. The output is then a combination of the long term memory state and the important parts of the input. But we will see this later on.

## Input and output sequences

Depending on the problem we are trying to solve we will need a specific structure of our network. For example, *sequence-to-sequence* RNNs are fed with a sequence of inputs, say N, and they return N outputs but shifted by one step in time.

In our case, from N inputs we would like the output of the next time step (we do not need the predicted outputs at every time-step). Therefore, we could use a *sequence-to-vector* network. That means, if we insert data for the last 2 weeks and our data is daily, we do not need the prediction that the network will obtain at every time step, we only need the prediction at day 2 weeks + 1.

A picture containing text, crossword puzzle, paper, handwriting

Description automatically generated

## Training of RNN

The main algorithm to train RNN is **backpropagation through time (BPTT).** Here is a schema about the process:

A picture containing text, handwriting, book, parallel

Description automatically generated

So a cost function is calculated using some of the outputs, not necessarily all and then the gradients of the cost function are propagated back to the beginning of the time sequence. For example, in our case, We could use a cost function that would be the mean absolute error between the probability of having a fire in that box and the truth value. We could calculate it only for the last output and not for all the outputs during the time series.

## Time series forecasting and implementation of a Deep RNN

#### Definitions

**Time series:** set of data that contains multiple variables for every time step

**Univariate time series:** set of data that contains just one variable for every time step

**Forecasting:** Using time series, predict future data

#### Size of input data

The shape of the input data is normally:

Dimensionality would be 1 for univariate time series.

#### Naïve forecasting

It is a simple and quick technique which consists of predicting the last value of a time series and evaluate its performance. It is good to compare our RNN with the basic models performance, because RNN can work very well if well implemented but very bad if not well implemented. Therefore, it is often good to have a reference of how good simpler models would perform, to see if we are actually taking the maximum potential of the RNN.

#### Trend and seasonality

The book says that when we have data with a small time scale but the data show seasonality at large time scales, then it is good to “take out” the seasonality to train the model, and then add seasonality once trained. I don’t know what that means exactly but we could be interested in it because our data is clearly seasonal: summer is the worst for wildfires.

#### Deep RNN

In deep RNN we have several recurrent layers between the input and the output. From now on, we will assume our networks is a Deep RNN.

## Forecasting several steps ahead

* If we want to predict only the data of, say, 10 days ahead with the data from the last 20 days, the method is easy: we just train the model using the data of 10 days ahead as the target data
* If we want to predict the data from the next 10 days, we have several methods:

1. First option is to use the model already explained: we feed the network with some inputs, predict the next step and then add this to the input data set to predict the following time step and so on. However, errors accumulate.
2. The second option is to train the model to predict the next 10 values at once. The structure will still be *sequence-to-vector* but the vector will not just have one probability for the following time step but 10 probabilities (probabilities of having a fire every day in the next 10 days)
3. The third option is to train the model to predict the next 10 time steps at every time step. Then we are turning it into a *sequence-to-sequence* network. We will have an output at every time step. This is a very stable way of doing it, because the backpropagation will not only occur at the last time step and propagate errors back in time. Backpropagation will occur at every time-step.

If one goes with this solution, all outputs are needed for the training of course, but the only important output for prediction is the one at the last time step. It is sometimes useful to use different metrics for evaluating the definitive output.

#### MC Dropout

The book gives advice on having some errorbars on predictions. I will read more about it later on.

## Tackling the Short-term memory problem

In RNNs, due to transformations of data, some of the information is lost at every time step. This is very inconvenient. It is like if we wanted to translate a sentence but when finishing the translation we don’t even remember the first words. To tackle this problems and have cells with higher memories we have more complex cells called LSTM Cells and GRUs.

#### LSTM Cells

They have a short memory state and a long-term memory state .

A picture containing text, map, diagram, schematic

Description automatically generated

The state c decides at each time-step which memories are dropped and which memories are added to the long term memory state. After that, the state c is passed trough an activation function, normally tanh and it produces the short term state h(t), which is the same as the output of this time step. More into detail:

* First the short term state of the previous time step and the input of the current time step are fed into four different simultaneous layers.
* Layer g(t) is the main layer and it would be the only layer present if our cell was not an LSTM cell. So its function is the usual: combine h(t-1) and x(t). The result of this, however, does not go straight to the output but some part of it will be dropped and some parts of it will be added into the long term state.
* The other three layers are gate controllers and they use the logistic activation function, so they output a 0 or a 1.
* The forget gate f(t) controls which of the elements should be dropped or not
* The input gate i(t) controls what part of g(t) should be added ot the long-term memory
* The output gate decided which part of the long term should be part of the output at this time-step.

#### Peephole connections

They can improve performance of LSTM cells, but not necessarily. The idea is that c(t-1) is passed as an input to the controller gates of input and forget. The current long-term memory state is used as an input to the output controller gate.

#### GRU cells

It is a simplified version of an LSTM cell and it seems it performs just as well.

A diagram of a cell phone

Description automatically generated with low confidence

The simplifications are:

* Both state vectors (short and long term) are merged into one h(t)
* The z gate controls both input and forget gates: If it outputs a 1, then the forget gate is open and the input gate is closed.
* There is no output gate: instead, there is a layer r(t) that decides which part of the previous state will be shown to the main layer g(t)

# End to end machine learning project checklist

#### Data

1. List the data we need and how we need it
2. Sample a test set, put it aside, and never look at it again

#### Data exploration

1. Create a copy of the data for exploration
2. Study each attribute and its characteristics: name, type, % of missing values, Noisiness and type of noise (stochastic, outliers, rounding errors), Usefulness for the task, type of distribution (gaussian,uniform…)
3. Identify the target
4. Visualize the data
5. Study correlations between attributes
6. Study how you would solve the problem manually
7. Identify promising transformations
8. Identify extra data that would be useful

#### Prepare the data

Advice; work on copies of the data

1. Data cleaning:
   1. Fix or remove outliers (optional)
   2. Fill in missing values
2. Feature selection (optional): delete attributes which are not useful for the task
3. Feature engineering, where appropriate:
   1. Discretize continuous features
   2. Decompose features ( for categorical data )
   3. Add promising transformations of features
   4. Aggregate features into promising new features (for example it might be better to use two combined features instead of using them separately)
4. Feature scaling: Normalize or standardize so that all data have the same scale