**Energy Forecasting Competition**

* Admin
  + 400 participating...200 watching
  + 80% no programming background
* Competition details
  + Wind energy
  + Wind farm
  + We will build the forecasting model, but trading model is given
  + Can write medium article after project
* Intro to Forecasting
  + Lead Time
    - How much into the future to forecast
    - T+5 eg
  + Measuring accuracy
    - Rmse, mae
    - Mae more common in finance cos of shocks in system more common
    - Can compare on if forecasting same thing and same time period
  + Persistence
    - T+x = T+0
    - Any credible forecast must beat persistence
    - Persistence is hard to beat for short lead times
  + Lag
    - Forecast with positive lag is useless
    - Hard to detect visually
    - Lag correlation coefficient
    - Lag of persistence is lead time
* Perceptrons
  + Order of trial of saturator
    - ReLu (fast)
    - Leaky relu
    - Tanh (long)
* Risk minimization
  + Batch learning
    - After getting loss from whole dataset, weights adjusted
    - One iteration is one run of the whole test dataset
  + What is used
    - Gradient descent to adjust weights after every iteration
    - Adam – deterministic, often finds good solutions, can get stuck in local minima
    - SGD – fast, works well on large datasets, may not give best solution
  + How
    - Back propagation
* Configurations
  + Initially can run lesser repeats, but later to confirm need to run more repeats
  + Might want to run more repeats for SGD solver
* Prediction pipeline
  + Preparation, windowing, training
* NNs are great at hierarchical clustering
  + HC is robust against noise
  + Hard forecasting problem: 1,4, 9,,16,25,36,? (cos cannot be reduced to a HCing task
  + It’s usually difficult to do HC in windowed data unless it is repetitive
* Normalization
  + Crucial for NN but not other ML techniques
  + Must use same normalization both in train and test
  + Preferably between [-1,1] because that is the region where learning will be able to occur rapidly with GD
* Network
  + Try 2/3 size of prev layer
* Difference network
  + Why
    - Many more examples of y-y0??
    - Risk and lag will be similar to persistence
    - Y-y0 is smaller so less need to normalize y (but do ti anyway)
* Overfitting
  + Bigger models tend to memorize training data
* 3 methods to overcome overfitting
  + Early stopping
  + Dropouts
    - Important
    - During training it randomly disables neurons below it
    - Does nothing in test
    - Dropout probability should be about 0.1, define this in config
  + Regularization
    - Adds an additional value to risk
    - Stick to L2
* Differencing
  + Notice that diff input works well in difference network
  + Idea: Use differencing in inputs where the difference is between values which are lead times away
* Momentum and force inputs
  + M(t) = xt – x(t-h)
  + F(t) = x(t) – 2x(t-h)+ x(t-2h)
  + Helps model detect movement and rate of movement and cluster accordingly
  + If you are predicting T+5, your momentum and force also has to be diffs by 5
* Input scaling and dimensional reduction
  + Problem
    - Unimportant input can slow learning
  + Input scaling subnet
    - Xnew = tanh(lam \* xold)
    - Clamping or symmetric squashing function (tanh) reduces outliers
      * Outliers are common in financial datasets, so leave it on unless you have a good reason not to
      * Clamping gave worse results in my toy problem
    - Lambda >= 0, prevent lambdas from cancelling out each other
  + Dimensional reduction (autoencoder) (compression/decompression subnet)
    - Cos NNs do not work well with large dimensional input (>100)
    - Autoencoder network
      * 3 layers
      * Tap bottleneck or center network as reduced version
      * Compression factor of 1/2 or more
      * You want to min reconstruction error
      * Why tanh?
      * The final LC layer must be the same size as input
      * A smaller bottleneck often leads to a larger error
  + Squared perceptrons
    - Train faster and learn better than ordinary perceptron
    - Do not use in preprocessing subnets
    - Y = theta(x.w + x^2.u + b), where w and u are to be learnt, and x^2 is just elementwise squaring
  + Momentum and force losses
    - Calculate 1st and 2nd order differences between predicted and actual
    - Ensure it does not dominate the main training loss
      * Bcos taking differences results in a noiser time series
    - Helps to reduce lag
    - M loss and f loss
  + Clustering
    - Regions close in input space have similar y-values
    - Although each layer has linear decision boundaries, as it gets mapped down to lower layers it gets nonlinear
    - In the last layer should not have too few neurons or cannot solve the problem?
  + VC (vapnik Chervonenkis) dimension

* Questions
  + Why plot actual vs predictions? How does this give lag? How is this better than the lag graph?
  + How did you come up force?
* Ideas
  + Use time 2d vector split
  + Forecast using 24 hours ago

**Results (Assignment 1)**

* Login
  + dld160
  + 558543
* Research
  + https://www.researchgate.net/publication/4871997\_Neural\_network\_forecasting\_for\_seasonal\_and\_trend\_time\_series
* Results
  + persistence-rmse= 0.11622133226936376
  + 0.0775549 nn8
  + 0.0769078 nn16
  + 0.0888164 nn32 A:0:-1 DIFF
  + 0.0600042 nn32 A:0:-5 DIFF
  + 0.0518136 nn32 A:0:-10 DIFF - really nice loss graphs
  + 0.0169951 nn32 A:0:-18 DIFF
  + 0.0170566 nn32 removed A:0:-5 SD (not a srong factor)
  + 0.0175881 nn32 A:0:-8 DIFF DIFF (added this but dosent help)
  + 0.010089 nn32 A:0:-24 DIFF (very good already, with lag 0)
  + 0.00993719 nn32 A:0:-30 DIFF
  + 0.00982033 nn32 A:0:-50 DIFF (extra features not helpful)
  + 0.00966342 nn64 A:0:-30 DIFF
  + 0.00948852 nn132
  + 0.0249092 nn64 SGD (losses are higher and more random, longer to converge, compared to Adam which is more stable, not helpful)
  + 0.00657269 redefine perceptron to innerpdt and relu (originally is tanh)
  + 0.00642003 lrelu = 0.001 (lrelu not helpful)
  + 0.00726686 lrelu = 0.2
  + 0.00604957 drop prob = 0.001 (best so far)
  + 0.00712983 drop prob = 0.01 (higher drop prob not helpful)
  + 0.00643496 L2 w\_decay = 10-4 (not helpful)
  + 0.00701755 L2 w\_decay = 10-5
  + 0.00640802 L1

**Assignment 2**

* Data
  + 30878 rows
  + E ile de frame
    - From 2017
      * Non0 from 2017,1,2,14
      * 7.48% 0’s
    - /1hr
    - In increments of 500
  + Wind/Weather forecasts
    - 8\*2 models
    - From 2017
    - Speed, direction (North) (6hr interpolated)
    - Anger vale 2 starts operating from 2 Jul
  + Forecast.csv updated daily forecasts from today to 10 days later (but you can only use T+48)
  + Historical is history also updated till yesterday
* Notes
  + It is more painful to over forecast
  + Since energy in units of 500, why not round the prediction to the nearest 500, do this in network.m
  + Todo
    - Momentum
    - Other techniques to reduce overfitting! IMPORTANT!
    - Dynamic clamp sizing
      * REMOVE NEGATIVE AND OVERLY POSITIVE PREDICTIONS
    - Maybe train on 18 data onwards
  + After submission
    - Draw Pnl in the end
    - 2020 persistence
      * I think persistence might be greater for 2020 as it is more erratic
    - Find unnormalized persistence and our loss
  + Others
    - ~~Energy values in caffe are not sorted (fixed)~~
    - ~~Look at weekly/monthly graph to see seasonality by year~~
      * ~~Can use T-256~~
    - ~~Maybe can use higher order interpolation for the wind data, bcos we are currently using linearly interpolated data every 3hrs~~
* Qns
  + Why need warmup
  + Is the energy production we are prediction across the whole of France
    - This is to see if the location of the wind forecasts is important
  + How do the different values of hours for interpolate affect the interpolation? Why need to interpolate, all the values for the forecast seem to be present?
    - Don’t need to interpolate in the end
  + What is temporal and spatial averaging?
    - Spatial is across different places
  + What is lab 4, about running pre.m offline?

**Comparing Loss functions**

* <https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/>
* <https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0>
* MSE (L2 loss)
  + Comparable to RMSE but don’t need to root as it takes time
  + Default for regression problems
  + Larger mistakes => more error
    - If outliers are important for business then this is better
* MSLE
  + Mean squared logarithmic error
    - MSE o5Log(x or x\_hat)
  + Use when predicting unscaled variables (large and small)
  + This penalizes less large difference in large predicted values
* MAE (L1 loss)
  + Used when data is largely gaussian but has outliers
  + Mae is robust to outliers
  + If you were to predict 10 observations, then use MSe you will predict mean, but using MAE median, and median is more robust to outliers than mean (eg 1,1,…1,1000)
  + Problem is (esp for NNs), is that the gradient is same throughout => grad large even for small loss values
  + To fix this we can use dynamic learning rate which decreases as we move closer to the minima (contrast to MSE)
* Huber loss (smooth MAE)
  + MSE when delta near 0 and MAE when delta large
  + Differentiable at 0 and less sensitive to outliers than MSE
  + Choice of delta depends on what you want to consider as an outlier
    - Might need to train hyperparameter delta iteratively
* Log-cosh
  + MSE when error is small and MAE for large error
  + Has all the advantages of huber loss
  + Is twice differentiable unlike huber loss
    - Many ML models like XGboost use newtons method to find the optimum where 2nd derivative is used

**FULL PIPELINE ON ML WITH DATA ANALYSIS**

* <https://towardsdatascience.com/machine-learning-with-python-regression-complete-tutorial-47268e546cea>

**ACF vs PACF**

* If dataset has a trend, you detred first, usu by one step differencing
* Use PACF to evaluate AR model
* Use ACF to evaluate MA model