MEX - COSMO Machine Learning Project

Bonvin Etienne, Pantet Xavier, Raynal Mathilde

***Abstract*—This report presents the work achieved by the team MEX in collaboration with the laboratory COSMO, within the framework of the class CS-433. The goal is to help in the process of finding the location of atoms in powdered solids by using electromagnetic shielding.**

I. Introduction

"NMR spectroscopy is a well-known and highly efficient method for probing the magnetic fields between atoms and determining how neighboring atoms interact with each other. However, full crystal structure determination by NMR spectroscopy requires extremely complicated, time-consuming calculations involving quantum chemistry - nearly impossible for molecules with very intricate structures." [1]. However the use of Machine Learning appears to be significant for this task and provides good results. Indeed physics phenomenons always follow rules that can be found in the underlying data. The COSMO lab already achieved an RMSE of 0.6 at project launch. They challenged us to find more efficient regression algorithms, or to try what neural networks could bring to the problem. Following the *No Free Lunch Theorem* [3], we aim to try as many methods as possible in order to find the best one for the given problem. The plan of this document follows the chronology of our work, starting with the data processing, then the machine learning algorithms we will try, to conclude with deep learning and the results.

II. Data Processing and Features Reduction

The data given by the lab has 30049 samples and 15961 features. This represents a lot and may slow down our computations or even worsen our results by introducing some noise. To reduce our feature space and filter the meaningful data, we combine different techniques to keep good predictions with less dimensions.

A. Correlation

We notice that the correlation is above 0.95 for some columns pairs. This high value shows an almost linear mapping. We made the choice to get rid of one of the two columns sharing a close distribution. Due to a complexity of $\mathcal{O}(n^2)$, we only manage to exploit the correlation between the 50th first columns, which as expected, does not affect the prediction a lot. We did not push this technique to its full potential as PCA presented next should theoretically take care of it.

B. Principle Component Analysis

To detect deeper relations between columns, we use a principle component analysis. We tried to use Maximum Likelihood Estimation to automatically find the optimum number of features, but due to the big amount of data, no result is outputted after several hours of run. We simply decide to try by hand some fixed number of components and see if they are a good fit. As we may see in the Fig.1 plotting the error given by a ridge regression with a train-test split of 75%-25% versus the number of components, we achieve one of the lowest RMSE when choosing approximately 3500 features.

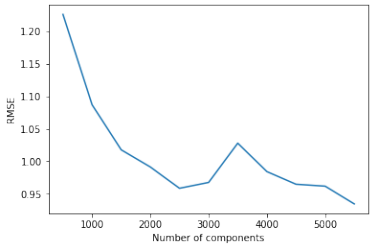

Fig. 1. RMSE vs number of components.

C. Data Augmentation

Deep learning and other modern nonlinear machine learning techniques rely a lot on the data at disposal. We can use jittering to create new samples from any sample. We define by jitter the noise that we add to our data, that we compute here as 1% of the mean of the feature. As long as this amount of jitter added to the new sample is sufficiently small, we can assume that the desired output will not change enough to be of any consequence, so we can just use the same target value [2]. We aimed for an augmentation of the number of samples by 10%.

D. Normalization

We make the choice to normalize our data by replacing every element of a feature column x_i by $(x_i - featureMean)/featureStd$. This guarantees stable convergence of weight and biases.

E. Whitening

This data processing does not show any noticeable difference but is aimed at ameliorating Neural Network results. It removes the underlying correlation in the data and provides unit component-wise variances. We chose not to use it as it did not improve our results in practice.

F. Results

In Table I are displayed the results of a ridge regression with an optimized regularizer and a polynomial expansion of degree one applied to the matrix processed with different techniques.

TABLE I
SUMMARY OF RESULTS FOR DIFFERENT TECHNIQUES

Full Matrix	Correlation	Jitter	PCA	Whitening
0.73	0.76	0.72	0.77	0.84

Using the same regression to every reduced matrix allows us to compare its "quality".

The final scheme that we applied to our data is the following:

1) Machine Learning
   a) PCA with 4500 features kept
   b) No whitening
   c) Normalization
   d) Jittering
2) Deep Learning
   a) PCA with 3004 features kept
   b) No whitening
   c) Normalization

The best predictions (presented later in the report) are achieved with a reduced matrix using 3004 features instead of 15000. We can see that among all the techniques we tried, a lot of them did not show any improvement for a fixed number of features.

III. MACHINE LEARNING

In order to meet the results obtained by the lab, we first try a few standard machine learning approaches. As already stated, our objective is a RMSE of 0.6, which is our first challenge proposed by the lab. With this in mind, we implement the following algorithms from the data matrix outputted from our feature reductions algorithms:

A. Least Squares

Least squares is a light weight solution to get a first hand-on the data. Using this method, we obtain a RMSE of 26, which is large, as we expected. Nevertheless, from this point, we find a first raw result, we ensured that our data can fit in memory and that we are able to handle it. Finally, we recall that least squares allows us to find the optimal solution (under well-known circumstances) to an optimization problem using MSE as loss function : $\mathcal{L}(w) = \frac{1}{2N}\sum_{n=1}^{N}(y_n - x_n^T w)^2$.

B. Polynomial Expansion

Then we move to the next level and decide to use feature expansion to decrease the error on our predictions. With polynomial expansion comes the question of the model complexity that we have to choose (the polynomial degree). We try degrees $1, 2, 3, 4$ in ridge regression (see below). It's worth noticing that from features $x_1, x_2, ..., x_n$, we get to a new feature space containing $1, x_1, x_2, ..., x_n, x_1^2, x_2^2, ..., x_n^2, ..., x_1^d, x_2^d, ..., x_n^d$ for polynomial expansion of degree d. No inter-features products are involved.

C. Ridge Regression

Ridge regression is an extension of least squares. More precisely, we add to least squares a special term (called regularizer) to measure the complexity of the model (here $\mathcal{L}_2$-norm) that prevents the model from overtitting. The loss function becomes : $\mathcal{L}(w) = \frac{1}{2N}\sum_{n=1}^{N}(y_n - x_n^T w)^2 + \lambda||w||_2^2$.

The hyperparameter λ allows us to decide how much importance we want to give to the regularizer and has to be tuned according to our needs. Moreover, we recall that this optimization problem has always an analytic solution for $\lambda > 0$.

D. Results

Using ridge regression with $\lambda = 10^{-i}, i \in \{-3, -2, ..., 18\}$, polynomial expansion with $d \in \{1, 2, 3, 4\}$ and 4-fold cross-validation we get our best results: RMSE of 0.71 (resp 0.74) for $d = 1$ (resp. $d = 2$) and $\lambda = 10^{-6}$. We deduce that adding a single constant feature allows us to go from RMSE of 26 to 0.7, which is a huge improvement. We can conclude that our data has a huge intercept. Moreover, we noticed that the test error is always the same (0.75) for any value of λ smaller than 1 and it starts getting larger and larger for $\lambda > 1$. We deduce that the model is not overfitted and that a regularizer is useless, which is consistent with what the lab told us.

E. Ideas pushed further

Since we have not reached our goal (RMSE of 0.6), we decide to try other losses and regularizers to see whether they could perform better than ridge regression on our dataset. In particular, we implement the following losses.

1) Lasso (MSE with $\mathcal{L}_1$-regularizer): The loss function of the lasso is the following : $\mathcal{L}(w) = \frac{1}{2N}\sum_{n=1}^{N}(y_n - x_n^T w)^2 + \lambda||w||_1$.

It is substantially the same than ridge regression except for the regularizer that is now a $\mathcal{L}_1$-norm. Also note that this function is not differentiable in $w = 0$. Therefore, SGD is required for optimization.

Before digging in depth of the method, we first try to set $\lambda = 0$ to run our SGD. We expect to reach RMSE of 26. In figure 2, we see that the convergence of our SGD algorithm is really slow. Moreover, we notice that the method is really sensitive to the value of the learning factor γ, making hyperparameters of the model really hard to tune. For these reasons, we decide to discard the lasso.

2) MAE: We also try the following loss function : $\mathcal{L}(w) = \frac{1}{N}\sum_{n=1}^{N}|y_n - x_n^T w|$.

Again, there exists no analytic solution to this optimization problem. Consequently we use SGD as well.

Figure 3 shows the evolution of the test error for $\lambda = 0$. The gradient descent is promising, so we decide to try different polynomial expansion degrees and different values of λ by adding a $\mathcal{L}_2$-regularization term in the loss function. We notice that the results are very close no matter the $lambda$. For for $\lambda = 10^{-5}$, we have a test error of 26.35 for a polynomial expansion of degree 1 and 24.17 for a polynomial expansion of degree 2.

Using this method we manage to get down to a RMSE of 24, which is not really conclusive.

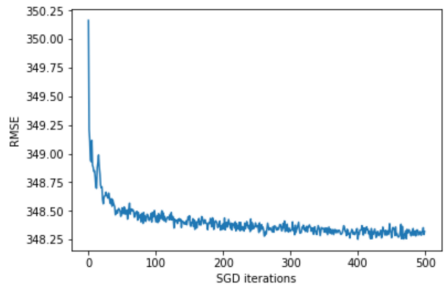

Fig. 2. SGD for lasso test error evolution.

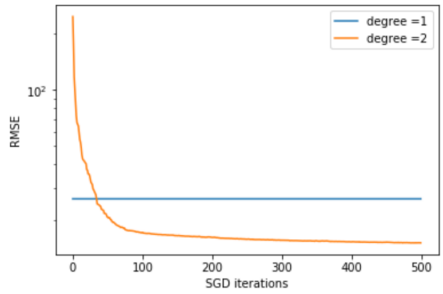

Fig. 3. SGD for MAE test error evolution.

F. ML conclusion

The best result we obtain is using a ridge regression, with an optimized regularizer and a polynomial expansion of degree 1. Our train and test errors are respectively 0.40 and 0.53.

We see that polynomial expansion of degree > 1 does not really help. Only adding a constant feature significantly decreased our RMSE, but we can imagine that adding cross-features products in our feature expansion could lead to better results. Without knowing more about the features involved, we could not do anything but trying all possible combinations of them, which was not doable in the given amount of time.

We know from the lab that MSE loss was well suited for this task for domain-specific reasons. Let's keep in mind that we started from a dataset of 15k features, which we had to reduce to process it. We have no specific knowledge for this task, hence it is possible that in the different dimensionality reduction methods that we have tried, we lost important part of information, preventing us to do better than RMSE of 0.53.

IV. Deep Learning

Standard machine learning already gives us good results. However we want to see whether a deep learning architecture reduces a bit further our error. This type of learning has been noticed to be often more efficient than standard machine learning mainly because of its flexibility and its capacity to capture the little relations between the features.

Note that the RMSE below are all computed for a 75/25 train / test split.

A. Fully Connected Neural Networks

Fully connected neural networks is one of the most known type of deep learning algorithms. It is in essence a polynomial regression made of neurons and weights.

To build our neural network, we first base ourselves on the tutorials from Tensorflow [4] and on the Standford lecture covering Neural Networks [5] in order to grasp the main components to optimize when constructing the network, which is the one that would fit our problem best.

The best neural network is rather large. The reason being that our problem is quite complicated. We have 15k features, 3k with the reduced matrix where each feature may have an impact, a correlation with all the others. We take care of the eventual overfit using regularization.

Then we execute a bunch of cross-validations to determine the best factors for the regularizer and the optimizer. See below the results of the cross-validation on the regularizer factor.

Regularization factor	10^{-1}	10^{-3}	10^{-6}	10^{-7}
Test error	.74	.49	.41	.43

The error stabilizes after 10^{-6} hence we decide to keep this value as it is the most restrictive one, encouraging the simpler models.

For the optimizer, we choose the Adam Optimizer [6] which has been reported to bring good results on data with a big number of rows and / or features, hence fitting perfectly our data. This optimizer is not going to change our result as long as it allows convergence and that it is not too slow. Hence we select a value of 10^{-3} which makes our neural network converge in less than 200 epochs.

B. Optimized structure

After cross validation on the different aspects of the neural network, the following gives us the best results for the best computation time :

- Train / test split : 75 / 25.
- Number of hidden layers : 8.
- Number of neuron / layers : 156.
- Activations : Rectified Linear Unit (relu).
- Regularizer : L2 regularizer with factor 10^{-6}.
- Optimizer : Adam Optimizer with factor 10^{-3}.
- Single network error : 0.43.

C. Dropout Layers and Max Norm Regularization

The above neural network gives us great predictions but we notice a significant overfit during training. We hence look into the litterature for ways to reduce this effect and enhance our predictions. In this area, dropout layers [7] combined with max norm regularizers [8] have been reported to bring very good results, especially when applied to big neural networks. However, as far as we pushed this idea, no better results than the one we already had were obtained.

V. Crowd of Neural Networks

For this approach, we base ourselves on the principle of *Wisdom of the Crowd* [9]. This principle may be derived concerning predictions in the following : "The average of the estimations of a crowd of non-expert people will be more precise than the estimation of an expert of the subject". This

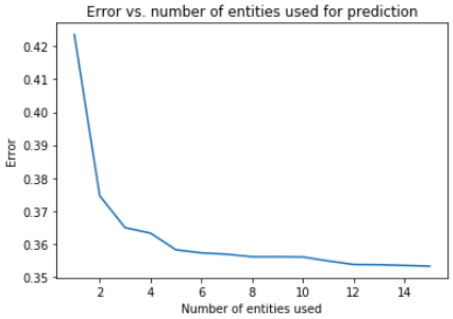

Fig. 4. Crowd error vs. number of networks.

principle holds as long as all subjects of the crowd do not give the same wrong answer.

Applying the above principle to our problem, we deduce that the average of a crowd of neural networks would give better estimations than a single highly trained neural network. Of course we also have to make sure that the elements of the crowd will not all give the same prediction which is ensured by the random aspect of the back-propagation phase.

A. Probabilistic Point Of View

We may think of the error of the prediction of a neural network as being a Gaussian distribution with mean μ and variance σ^2. Note that μ is not necessarily 0.

Let $N_1, N_2, ..., N_k$ be our neural networks where $N_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$. By averaging the predictions, we create a new random variable : $S = \frac{1}{k}\sum_{i=1}^{k} N_i$.

We can compute the expected value and the variance of this new random variable and their limit for $k \to \infty$:

$$\lim_{k\to\infty} \mathbb{E}[S] = \lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}[N_i] = \lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k} \mu_i = \mu_S$$

$$\lim_{k\to\infty} Var[S] = \lim_{k\to\infty} \frac{1}{k^2}\sum_{i=1}^{k} \sigma_i^2 \leq \lim_{k\to\infty} \frac{1}{k^2}\sum_{i=1}^{k} \max_{i\in[1,k]} \sigma_i^2$$
$$= \lim_{k\to\infty} \frac{k \max_{i\in[1,k]} \sigma_i^2}{k^2} = 0$$

Consequently, we create a model with variance as low as we wish as k grows large but still biased (see Figure 4). The bias is just the average of the individual biases of all the neural networks composing the crowd. However, this is already enough to significantly reduce our error.

B. Crowd Of Experts

The papers of A. Viikmaa [10] and the one of R.A. Jacobs et al. [12] propose to specialize the networks to perform well on a given subset of the data. The hard part of this task is to build a gating network which, for each input sample, should determine which neural network should be selected to make the best prediction.

We try two different methods to implement the gating network : one using a standard deep learning classification algorithm and the other one using k-means clustering. However, none of these two implementations give us better results than the ones we already have.

In R.A. Jacobs et al. paper, it is precised that specialization of the experts works well when applied to categorizable data. However none of our tests shown a clear type of categorization. This could explain the lack of results of our systems.

C. Collaborative Crowd

Continuing the idea of the crowd of neural networks, we think that the networks composing the crowd could learn from the errors of the other networks.

Hence we create a crowd where each network, instead of being trained only on the training matrix is also trained on the predictions of the previously added networks. That is $S_{train_k} = \{X_{train}|\hat{y}_0|...|\hat{y}_{k-1}\}$ where X_{train} is the training matrix, S_{train_k} is the training set of network k and $\hat{y}_i$ is the prediction of the network i.

D. Combination of Crowds

Our last idea is to combine crowds into a bigger one. The prediction of the crowd created this way will just be the average of the crowds that compose it.

E. RMSE of the different crowds

As a conclusion for this part, here are the RMSE of the different crowds over the test set.

Simple	Experts 1	Experts 2	Collaborative	Super
.353	.375	.377	.349	.342

Note that the crowds are ordered in the order of appearance in this report.

We observe that all the Crowds performed pretty well, however the **Composition of Crowds** (or SuperCrowd) is clearly the one with the smaller error but also with the biggest training and prediction time.

VI. Conclusion

This project is a very nice demonstration of the advantage of deep learning over machine learning when working with big datasets. The *COSMO* lab put a lot of hardwork into the generation of a matrix with a lot samples and many features, giving us a lot of meaningful data to work on.

Our feature reductions algorithms allowed us to significantly reduce our computation time while keeping good predictions. Then we used Machine Learning algorithms to the limit of our knowledge to observe that we couldn't do much better than a RMSE of 0.6. Finally, the deep learning models, perfectly fitted for the kind of datasets we had were impressively effective. With a few optimization techniques, we were able to obtain a RMSE of 0.342, almost half of our first objective.

The biggest impact of our project is that the lab now have empirical proofs that deep learning algorithms are better-suited and that they should continue digging in that area.

REFERENCES

[1] EPFL website of COSMO lab https://actu.epfl.ch/news/ai-and-nmr-spectroscopy-determine-atoms-configur-6/

[2] P. Koistinen, L. Holmstrom: Kernel Regression and Backpropagation Training with Noise https://pdfs.semanticscholar.org/489b/b5fdf3ae1163659fe45b38c2369f47dcbefd.pdf

[3] Wolpert, D.H., Macready, W.G. (1997) https://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf IEEE Transactions on Evolutionary Computation 1.

[4] Tensorflow Tutorials https://www.tensorflow.org/tutorials/keras/basic_regression

[5] Standford University CS231n Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/

[6] D. P. Kingma, J. Ba. Adam: A Method for Stochastic Optimization conference paper at the 3rd International https://arxiv.org/pdf/1412.6980.pdf Conference for Learning Representations, San Diego, 2015.

[7] N. Srivastava et al. Dropout : A Simple Way to Prevent Neural Networks from Overfitting *Journal of Machine Learning Research 15*, Toronto, Canada, published June 2014.

[8] N. Srebro and A. Shraibman. *Rank, trace-norm and max-norm.* In Proceedings of the 18th annual conference on Learning Theory, COLT05, pages 545560. Springer-Verlag, 2005.

[9] James Surowiecki (2004) *The Wisdom Of Crowds*, ISBN : 978-0-385-50386-0.

[10] A. Viikmaa : Combining multiple neural networks to improve generalization https://courses.cs.ut.ee/MTAT.03.277/2014_fall/uploads/Main/deep-learning-lecture-9-combining-multiple-neural-networks-to-improve-generalization-andres-viikmaa.pdf, University of Tartu, Estonia, November 2014.

[11] S. J. Nowlan and G. E. Hinton Evaluation of Adaptive Mixtures of Competing Experts http://www.cs.toronto.edu/~fritz/absps/nh91.pdf, University of Toronto, Canada.

[12] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton Adaptive Mixtures of Local Experts. http://www.cs.toronto.edu/~fritz/absps/jjnh91.pdf, University of Toronto, Canada Massachussets Institute of Technology, USA.