# IFT6135 - Assignment 1 - Programming

Salem Lahlou, Rémi Le Priol, Valentin Thomas
Monday February 19th

## Problem 1

### Building the model

We chose the architecture of the model as follows: we have 3 layers, first one has size $784 \times 512$, the second has $512 \times 256$ and the last $256 \times 10$. For short we will from now on write it as $\{784, 512, 256, 10\}$.

Counting the biases, the number of parameters per layer is :$\{401920, 131328, 2570\}$and in total: $535, 818$ parameters.

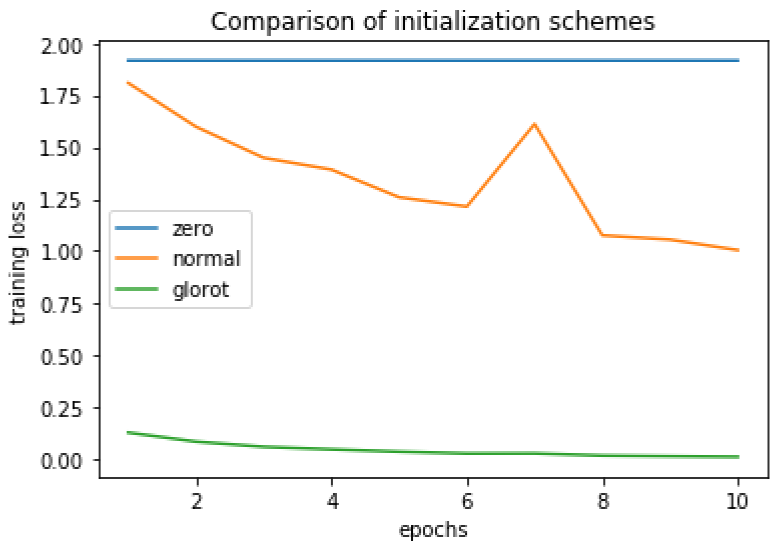
### Initialization

We report the training loss for the first 10 epochs with the following setup:

- Hidden units per layer $\{784, 512, 256, 10\}$for 535,818 parameters in total,
- ReLU non-linearity,
- learning rate = 0.01
- mini-batch size = 32

The results by initialization are:

- **Zero**: 1.92e+00, 1.92e+00, 1.92e+00, 1.92e+00, 1.92e+00, 1.92e+00, 1.92e+00, 1.92e+00, 1.92e+00, 1.92e+00
- **Normal**: 1.81e+00, 1.60e+00, 1.45e+00, 1.39e+00, 1.26e+00, 1.21e+00, 1.61e+00, 1.08e+00, 1.05e+00, 1.00e+00
- **Glorot**: 1.30e-01, 8.63e-02, 6.13e-02, 4.92e-02, 3.80e-02, 2.93e-02, 2.97e-02, 1.92e-02, 1.61e-02, 1.31e-02

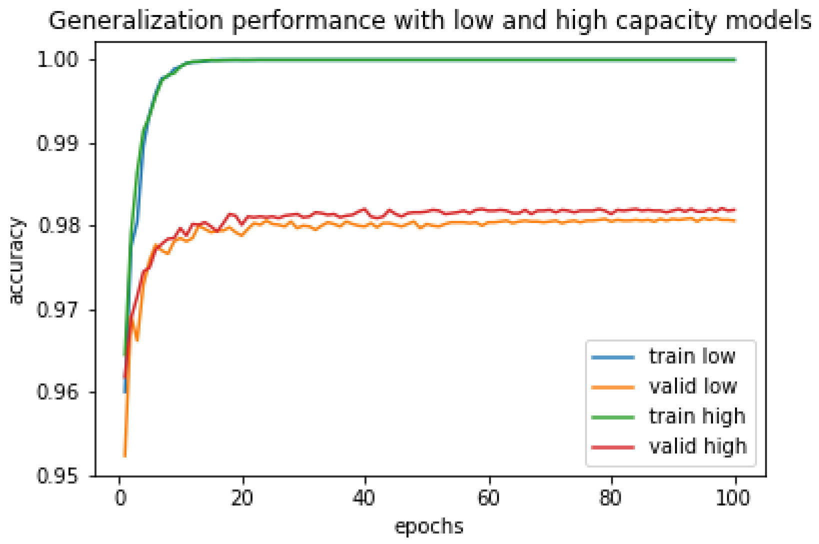


Initializing the weights to zero is a very bad idea. The network does not learn at all because the forward mode yield zeros everywhere, and so does the gradients in the backward mode. The normal initialization is already better, although the convergence remains quite slow. The Glorot uniform initialization is on a whole other scale of performance. After one epoch the score is already really good and it keeps improving with every epoch.

## Learning Curves

- Low capacity model : h = [784,512,256,10], total number of parameters = 535,818
- High capacity model : h = [784,1024, 512,10], total number of parameters = 1,333,770

We keep the ReLU, the learning rate 0.01 and the batch size 32.



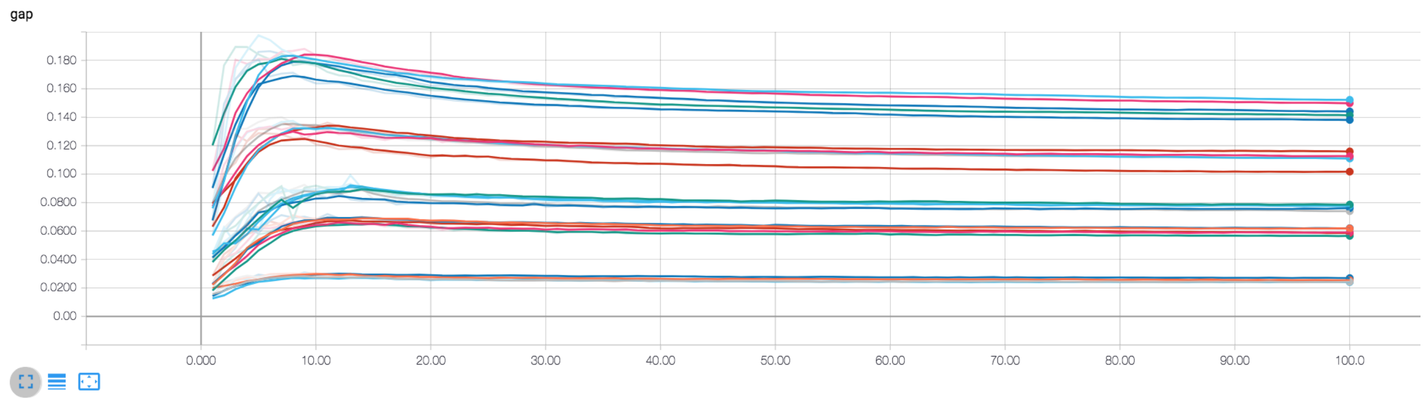
Both the low and high capacity model reach perfect accuracy on the training set after a quite low number of epochs. On the validation set this is a different story. We could expect the high capacity model to overfit and get decresing accuracy after a 100 epochs. This does not happen.

The validation set accuracy remains stable at about .98 for both models after 20 epochs or so. The high capacity model is marginally better than the low capacity model.

A possible interpretation is that increasing the model capacity lowered the bias without increasing too much the variance.

## Training Set Size, Generalization Gap, and Standard Error.

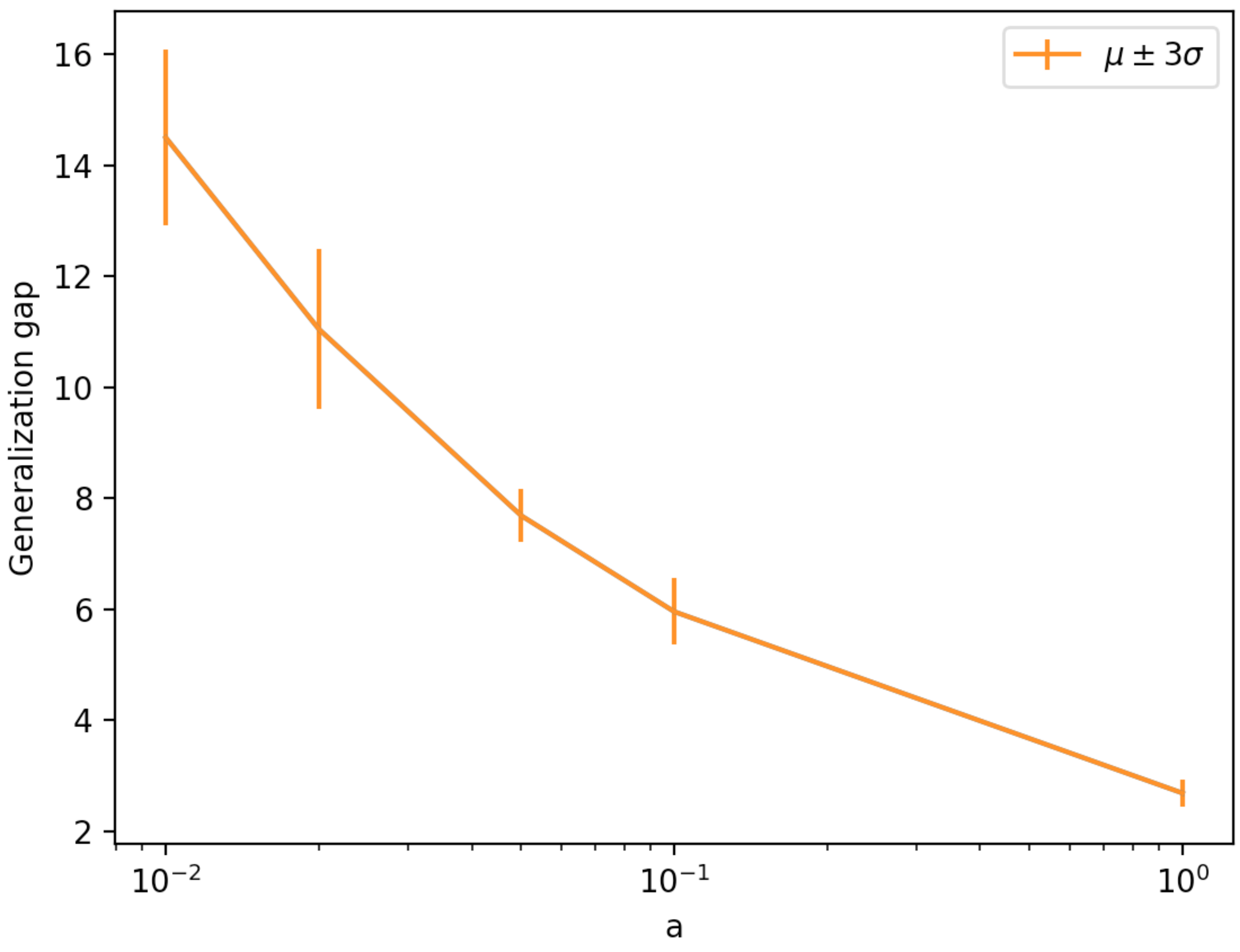
For this part, we used the small capacity model $\{784, 512, 256, 10\}$.



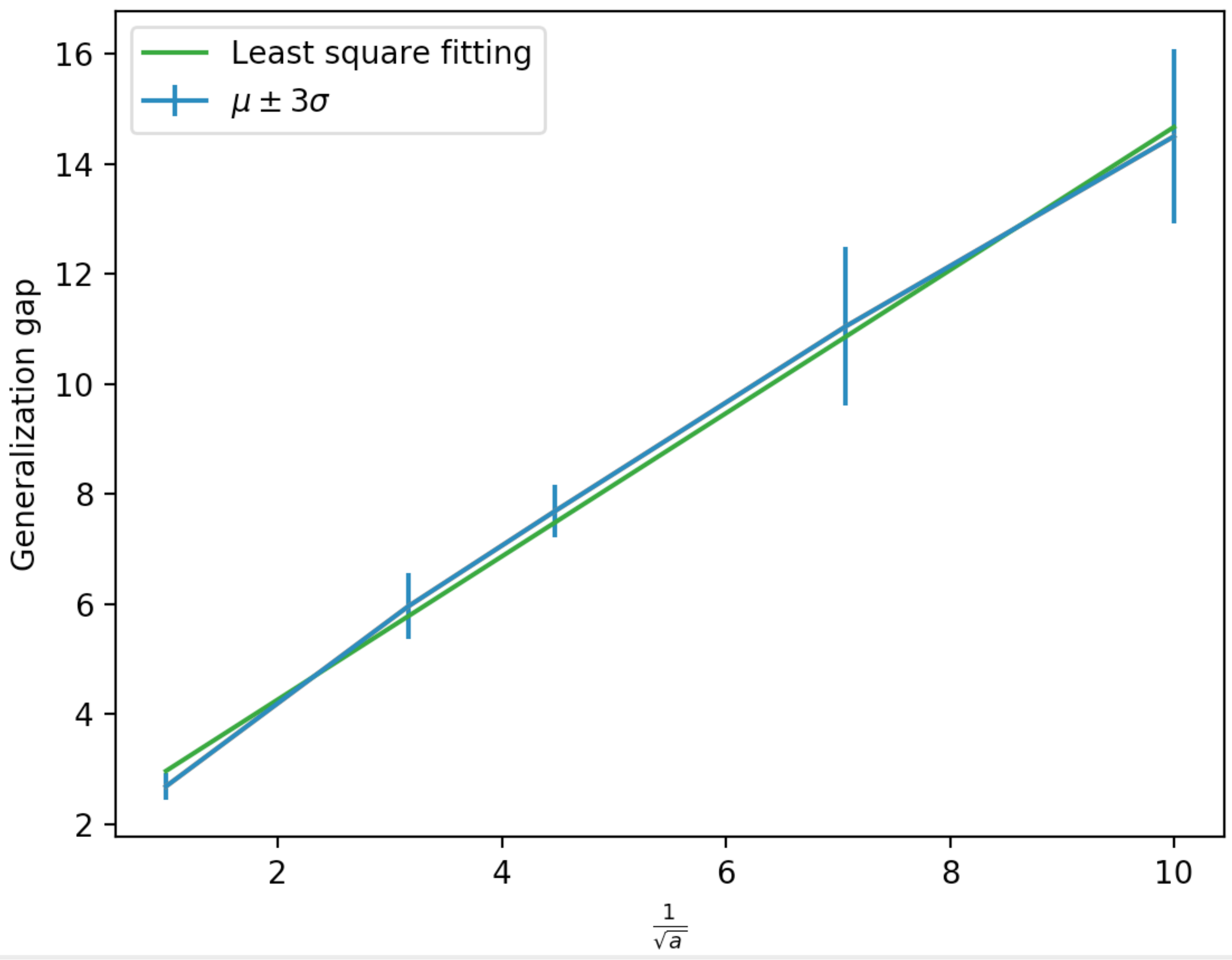
The generalization gap for the 25 experiments. We can see they are grouped by 5 which correspond to the 5 trials for each parameter $a \in \{0.01, 0.02, 0.05, 0.1, 1\}$

| **trial\a** | **0.01** | **0.02** | **0.05** | **0.1** | **1** |
|---|---|---|---|---|---|
| trial 1 | 15.22 | 11.27 | 7.86 | 6.17 | 2.46 |
| trial 2 | 13.80 | 11.09 | 7.78 | 6.20 | 2.54 |
| trial 3 | 14.97 | 11.60 | 7.40 | 5.90 | 2.68 |
| trial 4 | 14.40 | 10.17 | 7.64 | 5.86 | 2.59 |
| trial 5 | 14.12 | 11.13 | 7.75 | 5.67 | 2.65 |
| $\mu$ | 14.50 | 11.05 | 7.69 | 5.96 | 2.59 |
| $\sigma$ | 0.53 | 0.48 | 0.16 | 0.20 | 0.08 |

We observe that the generalization gap decreases (as well as its uncertainty) as we use more and more samples.



Generalization gap as a function of the sample size. Error bars are $\mu \pm 3\ \sigma$. Below is the same plot but as function of the inverse of the square root of the generalization gap. This is close from a straight line as the upper bound given by concentration inequalities as we can see from the least squares fit.

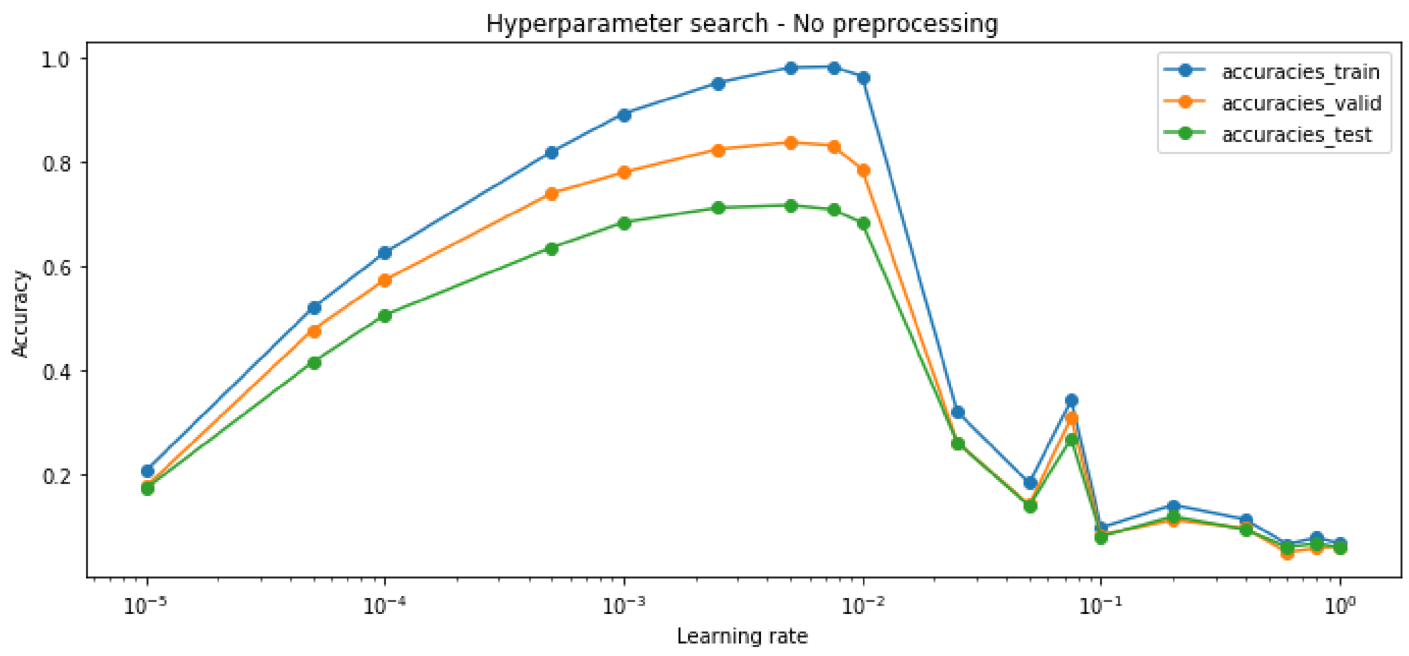


## Problem2

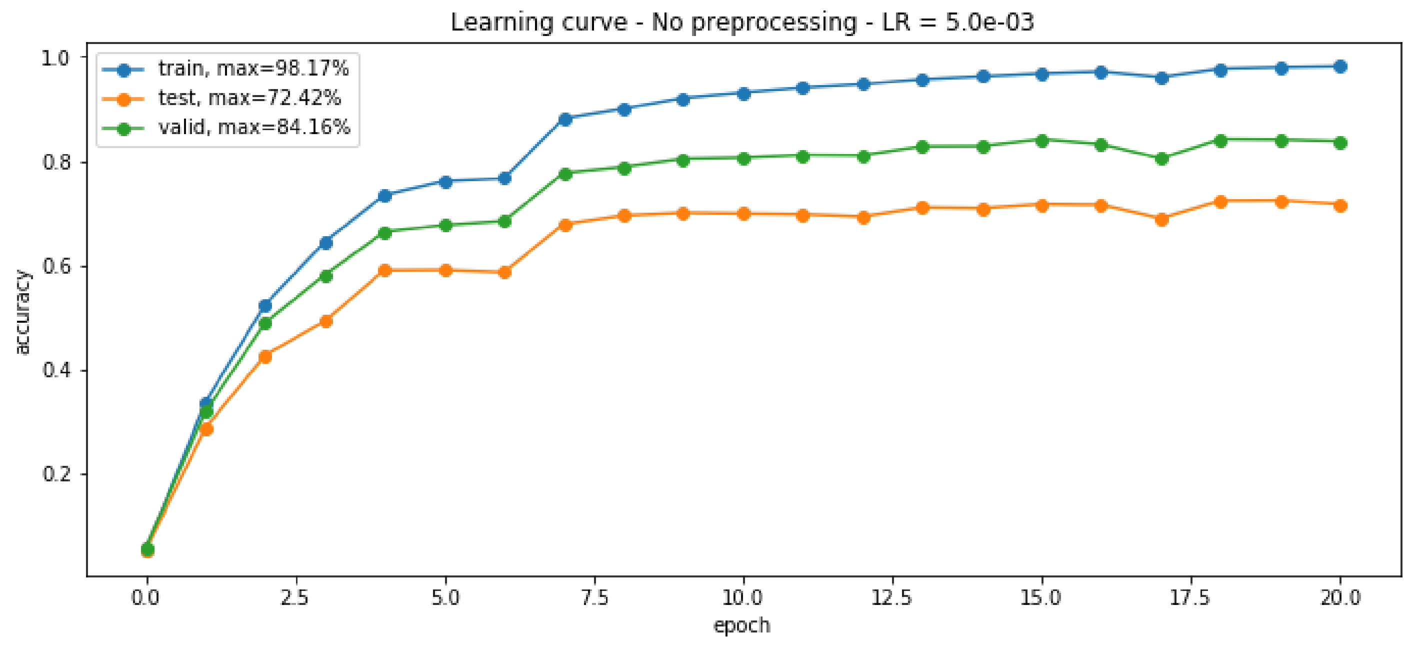
**1.** For each of the suggested pre-processing, we report the accuracy reached after 20 epochs for different learning rates.
We also report the learning curves when training with the learning rate that yields the maximum validation accuracy.
In all our experiments, we use a **mini batch size of 64, and the ReLu activation function**.
Here is a summary of the accuracy scores we obtained:

| | Train | Validation | Test |
|---|---|---|---|
| **No preprocessing** | 0.982 | 0.842 | 0.724 |
| **TF-IDF** | 0.999 | 0.901 | 0.799 |
| **Standardization** | 0.899 | 0.718 | 0.564 |

### No preprocessing



The training accuracy never reaches 1. Only small learning rates below 0.01 give good results. That's because the problem is so ill conditionned. Smaller learning rates ($< 10^{-3}$) perform badly because the optimizer needs more than 20 epochs for the learning procedure.



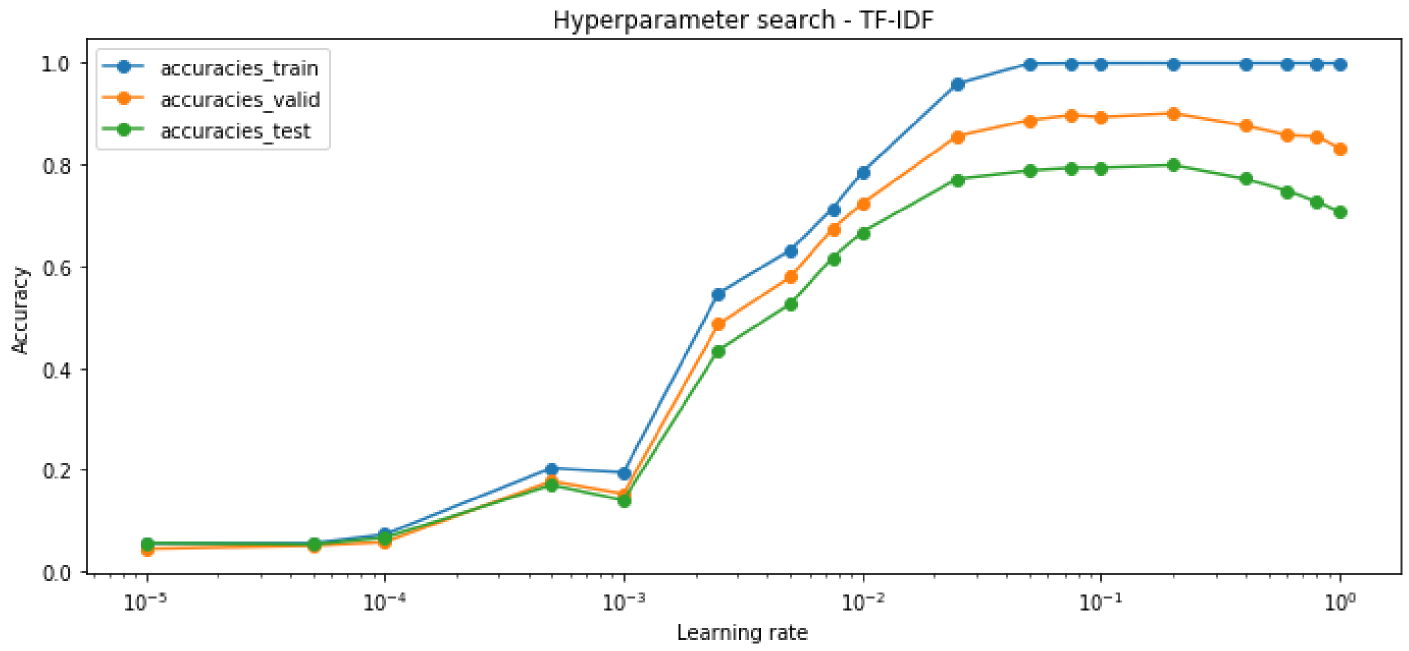
There is a gap between the validation and the test accuracies. At first, it may seem unusual given that the train/validation/test sets do come from the same distribution, but this gap makes more sense when we observe that:

- Among the 61188 words in the vocabulary, 10489 do not appear in the training set (this obviously may slightly vary given different train/validation splits of the original training set).
- 9545 words appear in the test set but not in the training set.

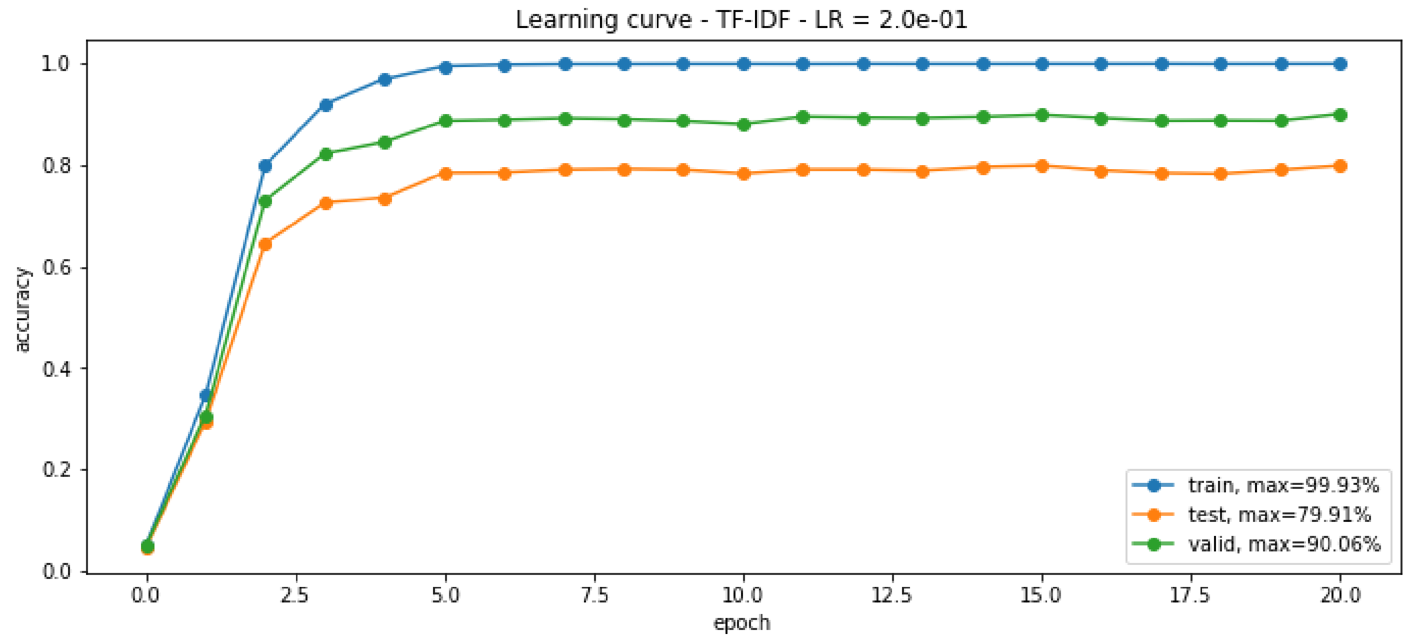
- 3276 words appear in the validation set but not in the training set (same remark as for the first point).

Thus, the model generalizes to the validation set better than to the test set.

### tf-idf

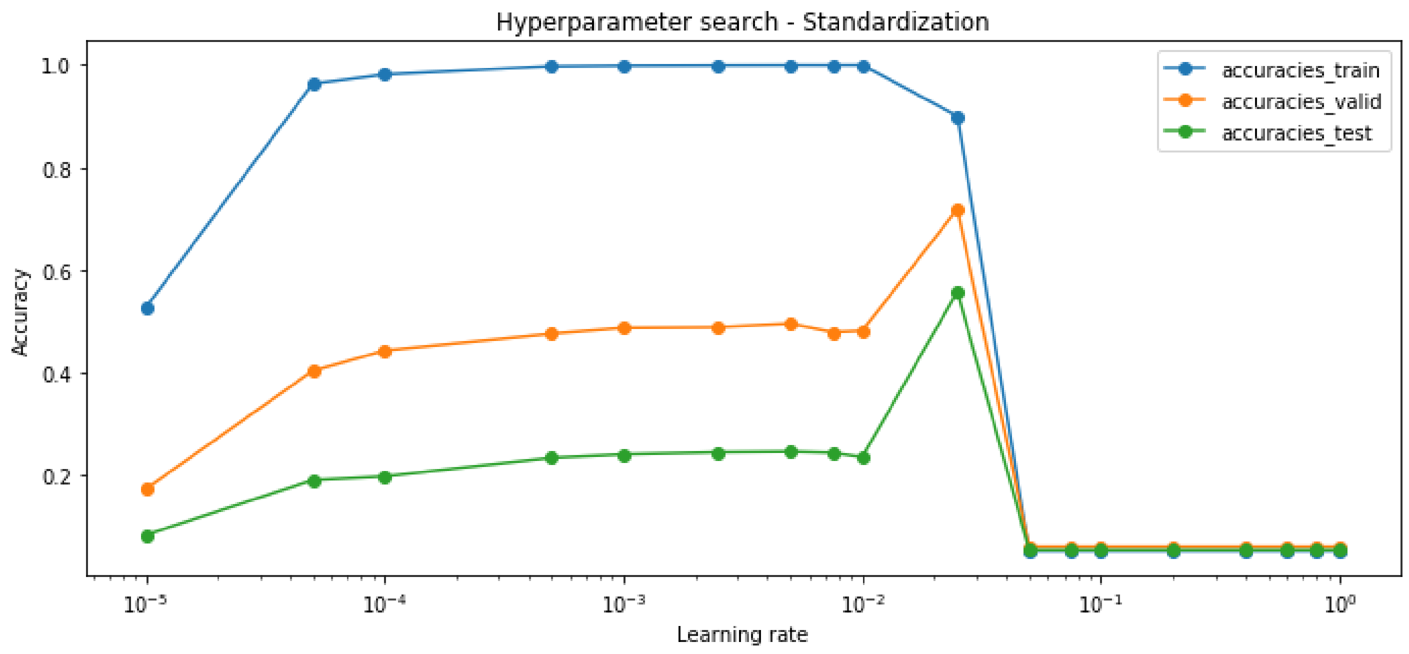


This time the problem is better conditionned, as the means of each word are closer to each other because reccurent words, e.g. stop-words, are scaled-down thanks to the IDF). Larger step sizes do the trick.



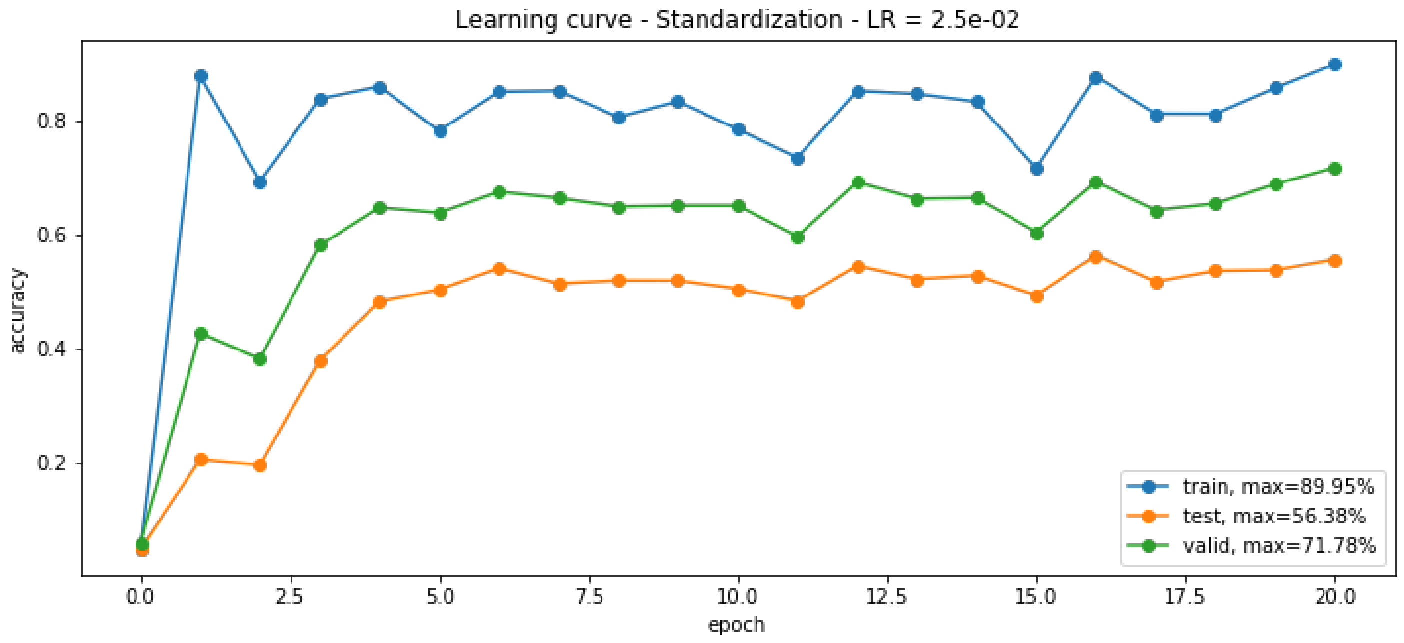
We see that there is a significant performance boost when compared to the model trained without preprocessing. The model also requires less epochs to achive a quasi-perfect score on the train set. This again can be explained by the well conditioning of the data, but also with the larger step size used here.

### Standardization



The model trained on a standardized training set has a hard time generalizing to the validation and test sets. This is mainly due to the fact that features (words) that do not appear in the training set take huuuuge values in the test and validation sets because of the division by $\epsilon$(variance is $0$). During training, the weights associated to these features (betweeen the first and the hidden layers) do not get updated, and their value is the one generated by the initilization scheme. And given that the activation function we chose is ReLu, at test time some neurons at the hidden layer have significantly higher values than those seen during training, thus yielding inaccurate predictions.
Obviously, some of the weights are not trained even with the tf-idf preprocessing (and without preprocessing). But the input values in the validation/test set are not huge, and the effects of

this “non-training of weights” is thus not as high. This is why the previous models generalize better.



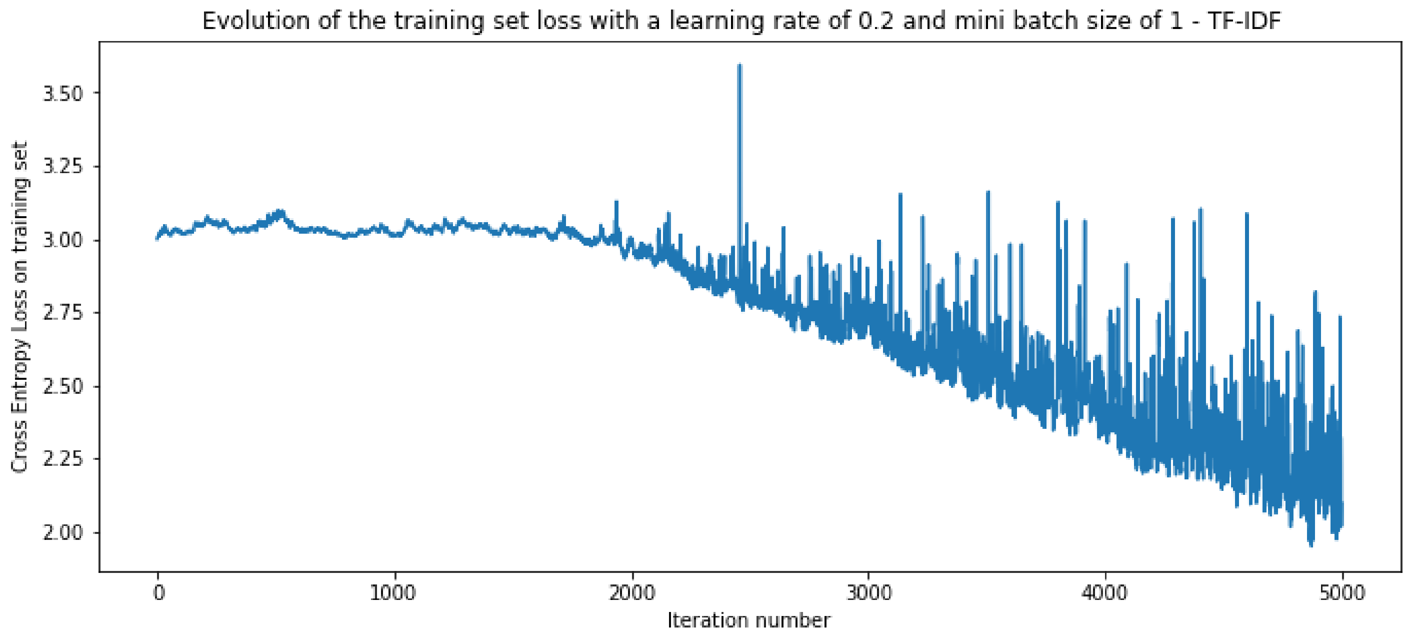
**2.**

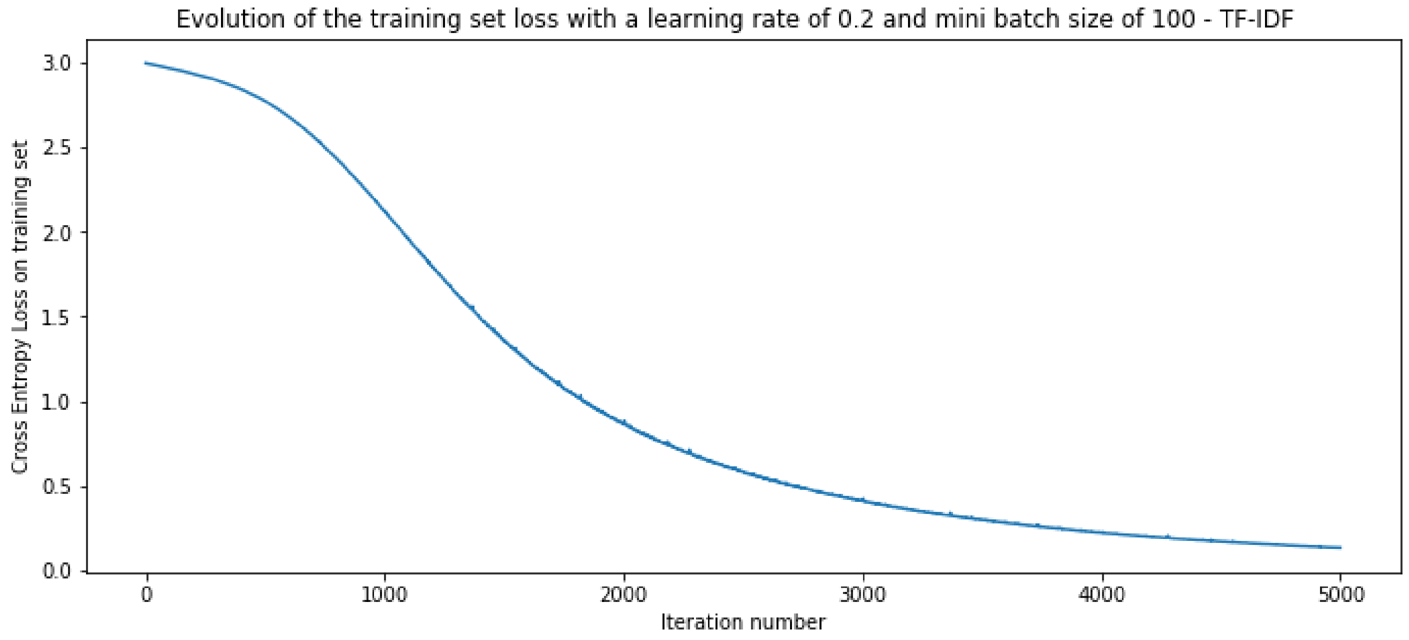
**2.**( a) No the same step-size can’t be used with every model. The reason is likely that some pre-processing give smoother, better-conditioned loss-surface than others. Namely tf-idf allow us to take large step-sizes of $0.1$, normalization can accomodate no larger than $0.01$ and no-preprocessing gives a very steep loss-surface for which step-sizes larger than $0.01$ don’t work.

**2.**( b) With $\varepsilon = 0$, the standardization wouldn’t work if a word had a 0 variance in the training set. In this case, we can simply delete that word from our dictionary (as in, do not consider it as a feature of the model). This cannot hurt the model because, if the zero variance is due to the fact that the word appears the same number of times in each document, then this word is non-informative and cannot be used to predict whatever we want to predict, and if the zero variance is due to the fact that the word is not present in the train set (which is our case here), then the learning algorithm hasn’t learned in any way how that word influences the predictions, and would keep some of the weights non-updated (as explained previously).

**2.**( c) The word count approach emphasizes documents that contain very common words (e.g. stopwords), without giving enough importance to more meaningful terms that appear more rarely. Thus, to give their true importance to these words, our model would need to put very large weights on them. This is quite hard and could require many gradients step. The problem is rather ill-conditioned. With tf-idf, the weights should all remain in a more reasonable range.

## Variance in training





**1.** The first thing we observe, is the difference in losses between the two models. While the model with a minibatch size of 1 requires 5000 updates to achieve a loss of $\approx 2.0$, the model with a minibatch size of 100 a $0.25$ after the same number of updates.
This can be explained by the number of epochs these 5000 updates run on. With a mini batch size of 1, the model does not even train on one whole epoch (the training set size is $\approx 9.5k$) for the first 5000 updates. This number of updates corresponds to $\approx 55$ epochs with the larger mini batch size, meaning that the model has learned more relevant information of the data during the loop through the epochs.
Secondly, with the small mini batch size, we observe a high variance along with the updates. This can be explained by the fact that when we compute the gradient on a mini-batch, we are actually approximating the gradient of the entire training set (used to perform the original batch gradient descent method). The gradient taken on a single data point is obviously a noisier estimation of the gradient than the one taken on 100 data points.

**2.** With small minibatch sizes, we can use momentum instead of vanilla SGD. It tends to accelerate SGD towards the right direction and diminishes the variance of the updates. The reduction of variance is due to the fact that momentum accumulates a moving average of past gradients and thus gives less importance to the new gradient evaluation, especially if it’s drastic enough to make appear huge variances as seen in the previous figure.