# Draft

Group DMML05: Nils Ziermann, Bjarne Gau, Colin Clausen
We have decided to do the following things:

- Ensemble of existing methods.
  We will create various ensembles from existing classifiers and experiment with leaving out single classifiers to measure their effect on the performance of the overall ensemble. The reason for trying this is because ensembles typically deliver better performance than single classifiers.

- SELU activated neural networks
  We will experiment with neural networks that use the SELU activation function (see https://arxiv.org/pdf/1706.02515.pdf). These promise to improve the performance of simple feeed forward neural nets without the use of convolutions or batch normalization.

- Random Search of Hyperparamters
  We will search for good hyperparameters in our experiments by randomly sampling hyperparameter configurations from reasonable ranges and evaluating them. This will be limited by our available computational power. This approach prevents us from having to come up with good hyperparameters for learning algorithmns that we have never seen before.

# First results

Our initial ideas were based on the example dataset that was very limited in size. This gave us the impression would could try a lot more computationally expensive methods than we actually could. The main problem was not the number of input documents, but the size of the input vocabulary. For the example dataset this was roughly 35000, which seemed managable. However this size turned out to be not static and grew with the dataset. The complete dataset uses a vocabulary of roughly 1.3 million words. This meant we had to adapt our suggested approaches.

**Ensemblemethods**

TODO

## SELU activated neural networks

The most simple type of neural network, the multilayer perceptron shown in Figure 1, uses fully connected layers. In these each unit is connected to all outputs of the previous layer. For the first layer this is the input of the network. For the complete dataset this poses a problem, as the size of the input exceeds 1.3 million datapoints. This means that first layer of the network would have 1.3

million weighted connections to each unit in the first layer from the input alone. Tests on the smaller dataset suggested a size of roughly 1000 units in the first layer would be a good idea, which however proved problematic to test within acceptable time, as the resulting network had nearly 1 billion parameters. The resulting network filled a large quantity of ram and a single fold took multiple days to complete on the server we were provided, putting over 12 cpu cores under full load.

Since the final results were intended to be tested with 10 fold cross validation this made proceeding further with this kind of network impossible. To proceed further with neural nets two issues had to be solved:

- The required computational power exceeds that of a CPU based server, a GPU based system promised speedups of 10 to 30 times.

- The networks memory requirements needed were too large to use the network on anything but the server with its immense amounts of system memory. No graphics card has the dozens of gigabytes the training run of the 1 billion parameter network required.

The solution to these two problems was found in the use of the LocallyConnected1D layer of the Keras framework, shown in 2. This is a layer that, similar to the convolutional layers, applies units that only have weighted connections to a few inputs at a time. However instead of the normal convolutional layers the weights are not shared. This effectively means the layers behave like fully connected layers with most weights cut out of them. By using locally connected units that are connected to 1000 input values each and are spaced out with a stride of 500 units the amount of parameters could be drastically reduced. Similar to convolutional layers the LocallyConnected1D layer also supports multiple feature layers all connected to the same output. This produces a 2D output that is then flattened again to be connected to normal fully connected layers. Preliminary tests showed no loss of classification performance, most likely since the input vectors are very sparse and high level connections between inputs far away from each other can be made in later layers of the network.

With the size of the network reduced to managable levels it became possible to run tests on a GPU instead of the CPU server, which speed up the experiments by nearly an order of magnitude.

## Results

The code used to created these results can be found on github: `https://github.com/ColaColin/Quadflor/blob/master/Code/lucid_ml/classifying/selu_net.py`. Training was done using early stopping based on using 9% of the training data of each fold as validation data. The validation criterion was the sample f1 score as calculated by sklearn. After 6 epochs of no improvements the learning rate was reduced by a factor of 10, after 9 epochs of no improvements trainings was considered to be complete. This typically happened after 80 to 120 epochs. Preliminary 1 fold experiments have shown that deeper networks

seem to not be very successful on this dataset. This suggests there may not be a big advantage of using the SELU activation function, as it is especially intended for deeper networks. The full 10 fold cross validation runs all were done with 2 extra hidden layers after the locally connected layer, so 3 hidden layers overall, as this seemed to provide good results.

**Network [10, 2048, 2048]**

| | |
|---|---|
| avg n labels gold | 5.240 ±0.015 |
| avg n labels pred | 5.410 ±0.245 |
| f1 macro | 0.223 ±0.006 |
| f1 micro | 0.505 ±0.009 |
| f1 samples | 0.508 ±0.005 |
| p macro | 0.245 ±0.012 |
| p micro | 0.497 ±0.017 |
| p samples | 0.555 ±0.008 |
| r macro | 0.231 ±0.005 |
| r micro | 0.531 ±0.009 |
| r samples | 0.525 ±0.009 |
| 10 fold cross validation run time | 36 hours |
| network size | 95.2 million parameters |

**Network [3, 1024, 1024]**

| | |
|---|---|
| avg n labels gold | 5.240 ±0.022 |
| avg n labels pred | 4.629 ±0.133 |
| f1 macro | 0.236 ±0.004 |
| f1 micro | 0.527 ±0.003 |
| f1 samples | 0.508 ±0.004 |
| p macro | 0.281 ±0.004 |
| p micro | 0.562 ±0.010 |
| p samples | 0.568 ±0.008 |
| r macro | 0.224 ±0.005 |
| r micro | 0.497 ±0.007 |
| r samples | 0.509 ±0.008 |
| 10 fold cross validation run time | 21 hours |
| network size | 22 million parameters |

**Network [2, 512, 512]**

| | |
|---|---|
| avg n labels gold | 5.240 ±0.023 |
| avg n labels pred | 5.932 ±0.596 |
| f1 macro | 0.230 ±0.006 |
| f1 micro | 0.500 ±0.013 |
| f1 samples | 0.488 ±0.010 |
| p macro | 0.248 ±0.015 |
| p micro | 0.474 ±0.033 |
| p samples | 0.493 ±0.025 |
| r macro | 0.241 ±0.007 |
| r micro | 0.532 ±0.015 |
| r samples | 0.544 ±0.016 |
| 10 fold cross validation run time | 22 hours |
| network size | 10.7 million parameters |

Table 1: Results of SELU networks. The first number of the network descriptions shows the number of feature layers of the locally connected layers. It can be seen that overall the middle sized network 3,1024,1024 performs best.

These values seem to be able to compete with the standard settings of the included classifiers, like sgd, however it is unclear to us how good they actuall are, as we cannot find the baseline for this specific dataset.
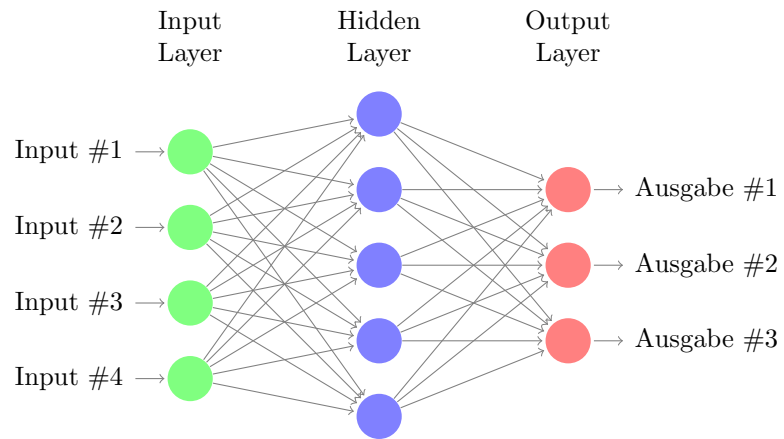


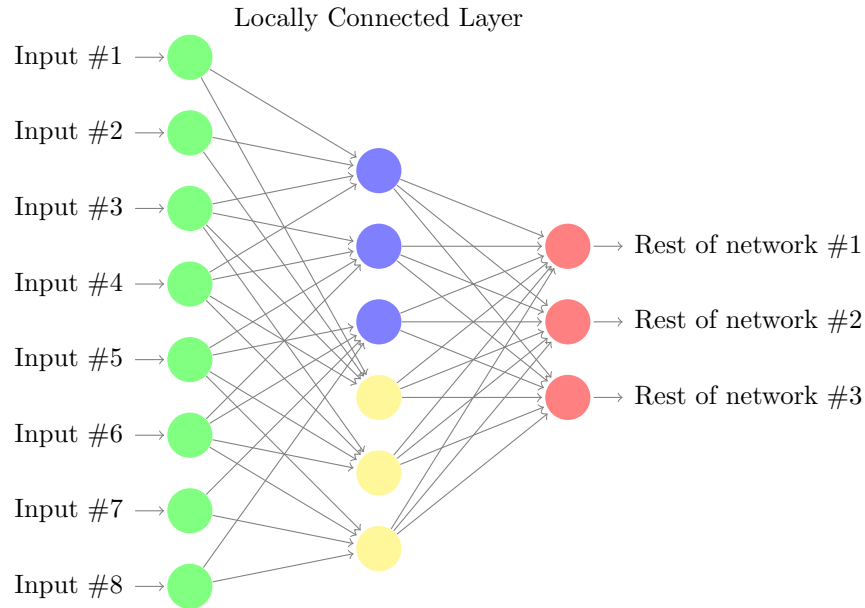Figure 1: A normal multi layer perceptron.

Figure 2: A simplified example of the locally connected layer. In the real code each hidden units is connected with 1000 input units at a stride of 500. This example connects each hidden units to 4 units at a stride of 2. In the real code a padding with zeros is used to make the first layer fit in. Notice that the locally connected layer in this example uses 2 feature layers, displayed in blue and yellow, that are all computed in parallel, similar to the feature layers in typical 2d convolutions, just in 1d.

## Random Search of Hyperparameters

The unexpectedly large number of vocabulary, especially on the complete dataset, has prevented us from doing any kind of random search. We've had to resort to guess a few promising values and try those. More computational power would be required to do a full random search.