Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1

Learning and generalisation in feed-forward networks — from perceptron learning to backprop

Maximilian Auer, Lukas Frösslund and Valdemar Gezelius

September 9, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were to familiarize ourselves and get comfortable with the detailed implementation of single-layer as well as multi-layer perceptrons for classification and function approximation tasks. Another objective was to understand not only the potential but limitations of various perceptron architectures, and learn how to minimize risk and allow networks to generalize well.

2 Methods

For the first part, Python 3 was used together with NumPy for mathematical operations and Matplotlib for plots. For the second part, the Python machine learning library Scikit-learn was used.

3 Part I

3.1 Classification with a single-layer perceptron

Classification with linearly separable data

Comparing perceptron learning with the Delta learning rule, we can see that the two algorithms converge similarly fast in terms of epochs. The main difference between the two in their respective convergence is that the Delta learning rule converges to a lower error (in this experiment, mean squared error was used). This makes intuitive sense since they differ mainly in that perceptron learning takes the binary response produced from the threshold output and computes an error, while the Delta learning rule uses the continuous value before activation to update the weights, enabling the updates to be more representative of the actual magnitude of the error.

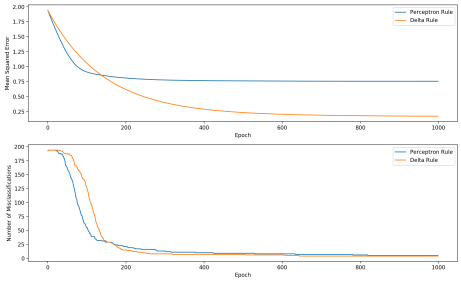

Figure 1: Comparison between Delta Learning Rule and Perceptron Learning Rule on Single-Layer Perceptron

When contrasting sequential with a batch learning approach, an apparent observation is that the sequential mode converges faster in terms of epochs. This seems logical given that it updates the weights more times in each epoch. Our experiments involved attempts to expose the vulnerabilities of the sequential approach. For example we tried not to randomly shuffle the input data, but there was no significant difference in performance.

Learning is generally sensitive to random initialization (of weights), because it can have a large impact on how many epochs is needed to converge. Using a sequential approach, random initialization becomes less bothersome because weight updates are more frequent.

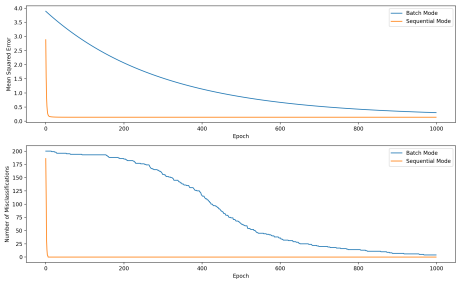

Figure 2: Comparison between batch and sequential approach for Delta Learning Rule on Single-Layer Perceptron

When removing the bias, our hypothesis stated that for a convergence with correct classification across all data samples, the two classes had to be distributed in such a way that a line crossing the origin could separate them. This seemed a sensible hypothesis, given that when bias is removed, the decision boundary will only be able to "twist" around the origin. By adjusting the data parameters and moving the distributions, we could confirm our hypothesis to be correct.

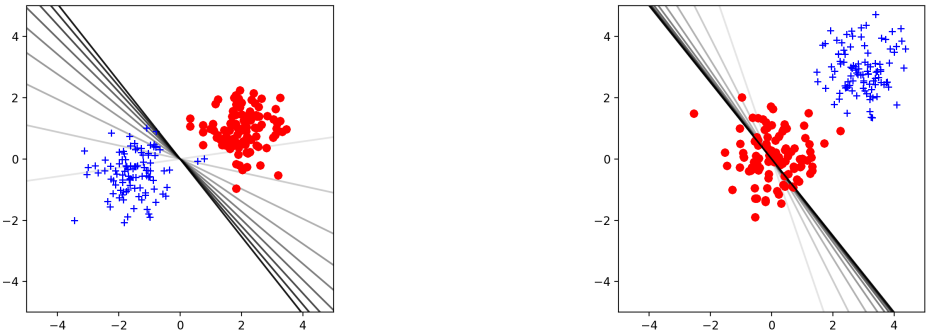

Figure 3: No bias. **Left:** data separable by a line crossing the origin. **Right:** data not separable by data crossing the origin.

Classification of samples that are not linearly separable

When contrasting perceptron learning against the delta learning rule in it's approach to non-linearly separable data, an obvious observation across several iterations of the experiment is that perceptron learning generally performs better in reducing the number of misclassifications, while the delta learning rule is more successful at lowering the average squared error. This makes sense based on the same rationale presented in section 3.1.2, in how the average error produced in perceptron learning is solely based on that binary thresholded output, hence the size of the error is in direct correlation to the number of misclassifications.

In our experiments with subsamplings, we chose to quantify the number of true positives and negatives respectively of the various subsamples. Across several iterations of the experiment, we could first observe that the complete dataset generally performed better than any of the subsamplings. Another clear conclusion was that subsampling severely affected the minority class, the class with fewer training examples, in a negative way, and that uneven class representations generally resulted in worse generalization.

It's no surprise that uneven class representation affects generalization negatively. Especially interesting is the minority class, because this class may be neglected in the training process. A non-representative sample distribution would similarly affect the generalization capability in a negative way.

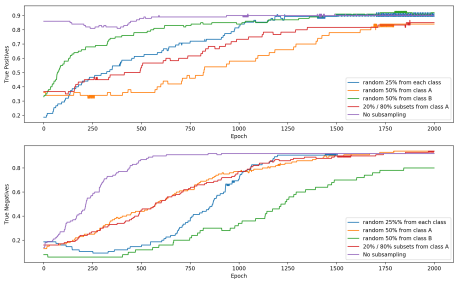

Figure 4: Comparison between different types of subsamplings on non-separable data

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data

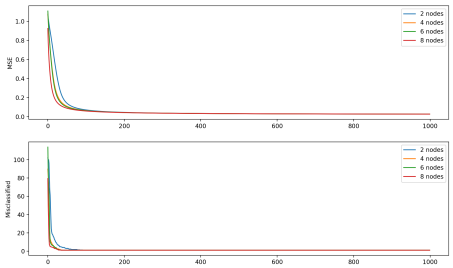

Figure 5: Comparison between networks with different number of nodes in hidden layer

1. We created neural networks with 1, 2, 4 and 8 nodes in the hidden layer. With 1000 epochs and η set to 0.004, we got (given the distribution of the the two classes) that none of the networks could fully separate the data. We could not find significant differences between the networks performances. The average misclassified points was 25.2, between the different node setups, with a standard deviation of 2.24 points and the average MSE was 0.394 with a standard deviation of 0.0085.

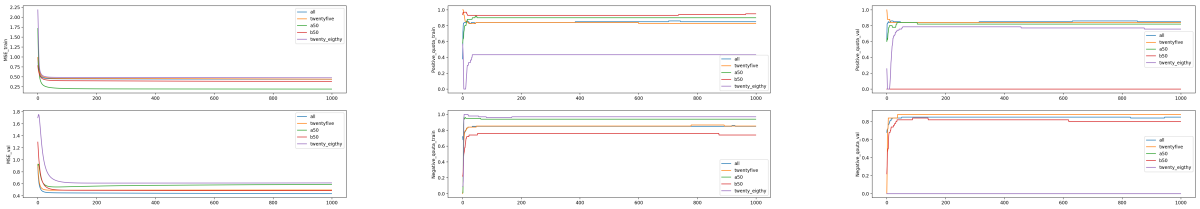

Figure 6: Sequential learning, imbalanced/balanced data, training vs validation, 8 nodes in hidden layer **Left:** Mean squared error for training and validation sets **Middle:** Positive- and negative class classification quota, training **Right:** Positive- and negative class classification quota, validation

2. Running 1000 epochs with η set to 0.004, we conducted tests for 1, 2, 4, 8 and 16 nodes in the hidden layer. The conclusions that could be drawn was that we got the lowest MSE for a configuration with 8 nodes, displayed in Figure 6. Concerning the different class imbalances, we saw that as expected, the setup when we removed 25% of each class for validation got the lowest validation error[1]. Conclusions from the classification graphs in Figure 6 include: when we remove data points from a specific class, creating an imbalanced dataset, the network has a harder time classifying points of that class and with a balanced dataset the both quotas are balanced out, the network strive for balance. For the batch learning, we see that it overall takes more epochs to converge since we update the weights less often. This leads to a flatter rise of the validation curves for the classification quotas.

[1]the configuration *all* in Figure 6 uses all data for both training and validation

3.2.2 The encoder problem

Our implementation seems to converge when we find a *"sweet spot"* for our value on η and the spread of the initial weights, given we run a fair amount of epochs, 10000, since our data-set only contains 8 data points. We seem to get better encoding for higher values of η, 0.007 with a spread of 0.7.

If the input is a 2^n vector, the compressed representation has to be represented by at least n bits. Since the input can be viewed as an one-hot 8-bit vector, the code (the activations of the hidden layer) of the encoder represent a 3-bit compressed version of the input vector.

When 2 nodes are used in hidden layer we could not get the network to classify every pattern correct, due to the argumentation in previous section.

Autoencoders could serve the purpose of finding a low-dimension representation of the data set it trains on, this could be used in different contexts, e.g information retrieval, anomaly detection and image processing.

3.2.3 Function approximation

We choose to calculate the error for this task as the MSE between the correct function output of $f(x, y) = e^{-(x^2+y^2)/10} - 0.5$ and the approximated output of our network. We saw that after 20 around hidden nodes, there wasn't a significant decrease of the error when increasing the number of hidden nodes. As seen in the figures below, a single hidden node creates a flat plane, 2 hidden nodes create a ridge, and 3 hidden nodes creates a triangle shaped hill. We can also see that the more hidden nodes we have, the better approximation we get. Even though we get a more exact result, the performance gain is minimal above a certain model complexity, especially considering the increased computational and time cost.

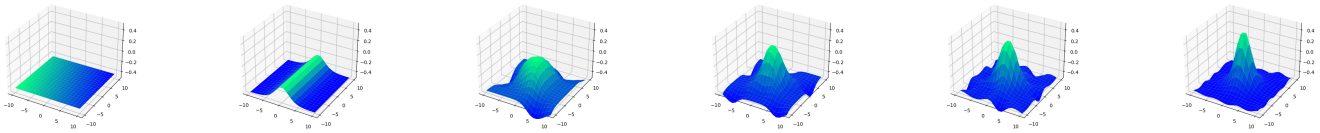

Figure 7: Approximations with different number of hidden nodes From left: 1, 2, 3, 5, 10, 20 hidden nodes

As we can see in the graph below, the error is not decreasing with a significant amount after around 15 to 20 hidden nodes. Therefore we choose to use 19 nodes as our most efficient model for the subset-test as it had about the same error as the models with higher number of hidden nodes.

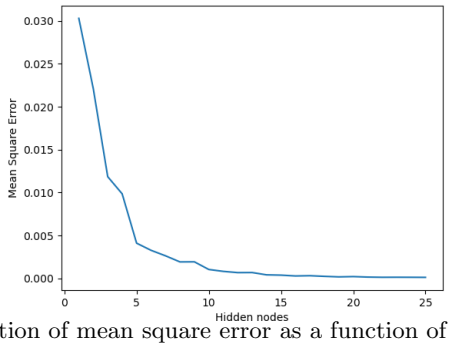

Figure 8: Approximation of mean square error as a function of number of hidden nodes.

Looking at the table below we can see that by using a random subset of the data we get a better performance with a bigger training set. The average MSE for the full set was 0.000187 which we can use as a comparison for these values. The generalization was therefore penalised by faster convergence. Selecting a subset in a non random manner lead to better generalization than the random approach, for example removing every other data point preventing data being unevenly spread.

Table 1: Validation MSE of different sub set sizes. The mean, standard deviation and difference between full data set mean.

%	Mean	Std	Delta Mean
0.2	0.005678	0.001776	0.005491
0.3	0.004086	0.001355	0.003899
0.4	0.002580	0.000712	0.002393
0.5	0.002007	0.000561	0.001819
0.6	0.001218	0.000394	0.001031
0.7	0.000893	0.000430	0.000706
0.8	0.000465	0.000230	0.000278

4 Part II

4.1 Two-layer perceptron for time series prediction - model selection, regularisation and validation

A neural network was trained on different combinations of nodes in the hidden layer (1 to 8) and strength of regularization (0.00001 to 0.1). L2 regularization was used, optimized by SGD and a momentum of 0.9. The validation split was 0.1 of the training data. A learning rate value of 0.01 was used, together with an early stopping regularizer. Mean squared error was calculated and averaged across 10 iterations for each of the 40 different combinations of regularization strength and size of hidden layer.

When analyzing the distribution of weights for different regularization strengths (Figure 9), we could see that a higher strength number of regularization decreases the variance in weights and pushed the values closer to zero.

Best performance (lowest MSE on hold-out set) was generally achieved on combinations with fairly low regularization strength and higher number of nodes. A network with 6 hidden nodes and a lambda value of 0.0001 was chosen to be the most promising combination. Conclusive evaluation can be seen in Figure 10, a comparison between the predictions on an unseen test set and the true time series samples.

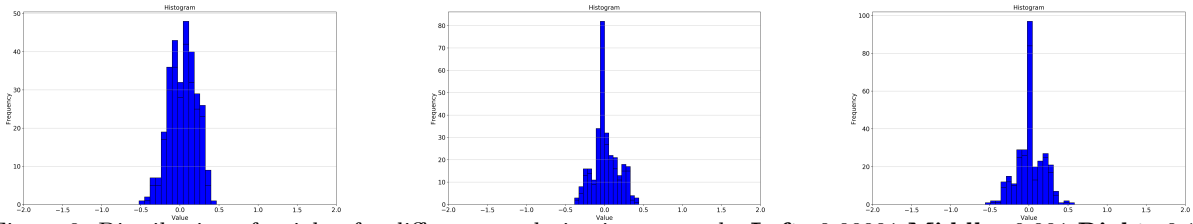

Figure 9: Distribution of weights for different regularization strengths **Left:** 0.00001 **Middle:** 0.001 **Right:** 0.1

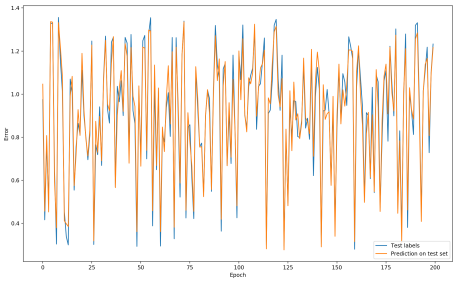

Figure 10: Final evaluation for two-layer time series analysis. Comparisons between predictions and ground-truth.

4.2 Comparison of two- and three-layer perceptron for noisy time series prediction

Validation prediction performance, measured by MSE loss on hold-out set, was calculated for all combinations of second hidden layer nodes and second moment of the additive gaussian noise. A mean was drawn from 20 iterations of each experiment. Regularization strength and learning rate was fixed at 0.0001 and 0.01 respectively. The loss tended to decrease with a higher amount of nodes, and in general there were no conclusive findings that separated how the losses behaved depending on the amount of additive noise, although there was a slight tendency of increased volatility for higher amount of noise.

When varying the regularization strength, we can see in Figure 11 (right) that, while not very conclusive, less noise tends to yield a lower validation loss, especially when the strength of regularization is low. Our findings indicated that for all levels of noise, the loss follows a similar curve when the strength of regularization varies.

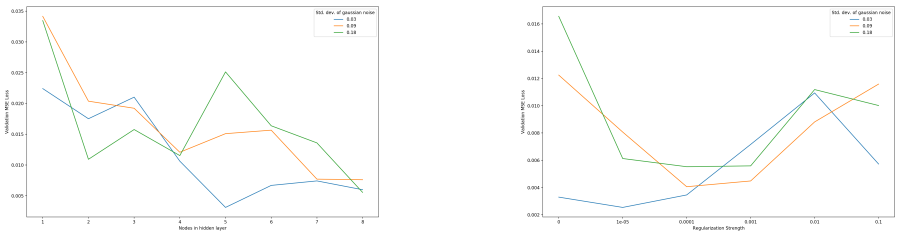

Figure 11: Experiments on three-layer time series analysis. **Left:** Varying number of hidden nodes **Right:** Varying regularization strength

The computations can be divided into three main parts. The forward pass, the backward pass and the weights update. The forward pass and the backward pass are increased equally when scaling from a two-layer to a three-layer network. Basis of the computational cost is essentially the number of operations performed in the matrix multiplications, which depends on the number of layers (i.e. the number of matrices) and the size of the hidden layers (i.e. the size of the weight matrices). For the weight updates, one operation is performed for every weight in the network. Let us assume the following:

$n = number\ of\ training\ examples$
$i = number\ of\ nodes\ in\ input\ layer\ (for\ us\ it\ is\ 5)$
$j = number\ of\ nodes\ in\ first\ hidden\ layer$
$k = number\ of\ nodes\ in\ potential\ second\ hidden\ layer$
$l = number\ of\ nodes\ in\ output\ (for\ us\ it\ is\ 1)$

For the forward pass and backward pass, the difference in computational cost between one and two hidden layers can be illustrated by following:

One hidden layer: $O(n(ij + jl)) = O(nij) + O(njl) = O(nij) + O(nj)$
Two hidden layers: $O(n(ij + jk + kl)) = O(nij) + O(njk) + O(nkl) = O(nij) + O(njk) + O(nk)$

And for the weight updates:

One hidden layer: $O(ij) + O(jl) = O(ij) + O(j)$
Two hidden layers: $O(ij) + O(jk) + O(kl) = O(ij) + O(jk) + O(k)$

Here we see how the computational cost scales with the number of hidden layers, and additionally we can see how it depends on the number of nodes in each of the hidden layers (j and k).

5 Final remarks *(max 0.5 page)*

- The lab was too exhaustive, could have been more focused.
- Some of the outcomes were quite inconclusive, especially when comparing the strength of certain hyperparameters and regularizers.