MSE | MASTER OF SCIENCE IN ENGINEERING

TSM – Machine Learning

PW 10: Artificial Neural Networks

27 November 2023

Daniel Ribeiro Cabral – Bastien Veuthey

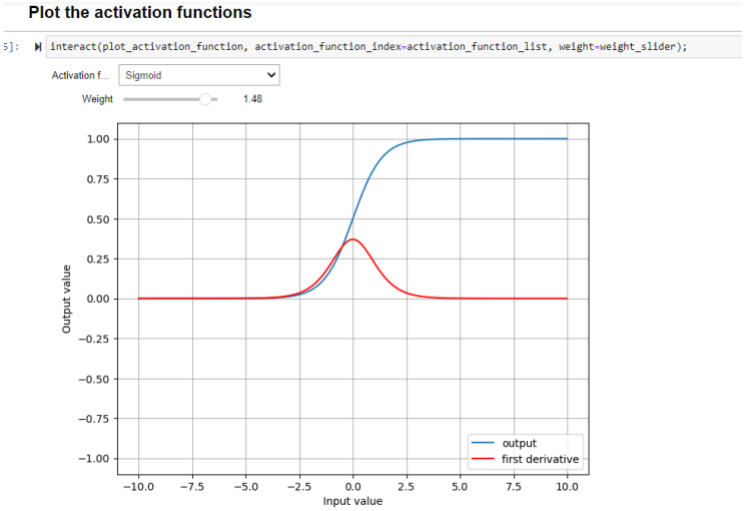
MSE | Data Science and Computer Science

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

The Perceptron and the Delta rule

1_activation_function

1. Observe the shape of the different activation functions proposed.



2. Observe the effects of modifying the weight. How the shape of the function changes ? How the first derivative changes ?

- **« linear function »** : The linear activation function is a straight line with the slope equal to the weight of the connection. Changing the weight changes the slope of the line. The derivative is constant and equal to the weight.
- **Sigmoig activation** : It's an S-shaped curve that outputs values between 0 and 1. Increasing the weight makes the transition from 0 to 1 steeper, effectively making the neuron's activation more sensitive to changes around the input of zero. The derivative has one peak and then comes back to zero.
- **Hyperbolic Tangent** : This is also an S-shaped curve but outputs values between -1 and 1. Similar to the sigmoid, increasing the weight makes the curve steeper around the origin. The derivative's peak becomes higher and narrower similar to the sigmoi. The peak is much bigger at weight of 2.00.
- **ReLU** : It outputs the input directly if it's positive; otherwise, it outputs zero. : Changing the weight will scale the output linearly for positive inputs. The derivative is either 0 for negative inputs or equal to the weight for positive inputs, indicating that there is no gradient for negative inputs.

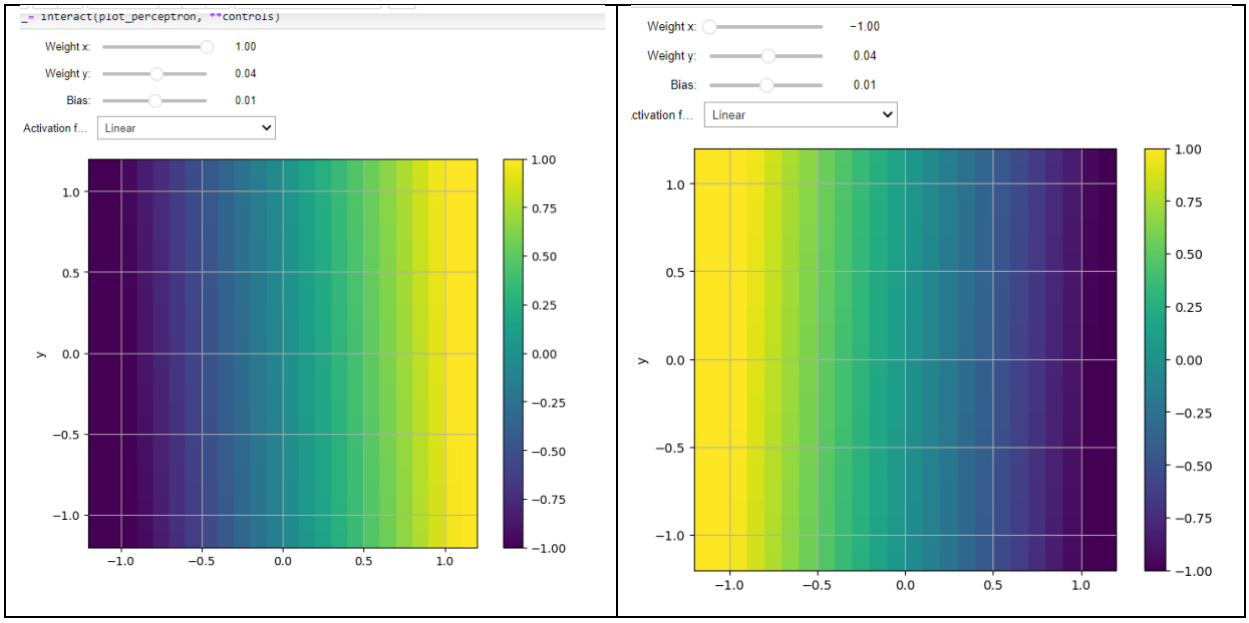
2_ perceptron

Select different activation functions and observe the output of the perceptron for different weight configurations

Here is what i can observe from the different parameters that we can observe from the graphs. Each having almost the same function in each activation function.

- **Changing weights** : Adjusting weight_x and weight_y will change the orientation and shape of the decision boundary that the perceptron learns to separate different classes or predict values. Hre below is an exmple oft he weight x with values 1.00

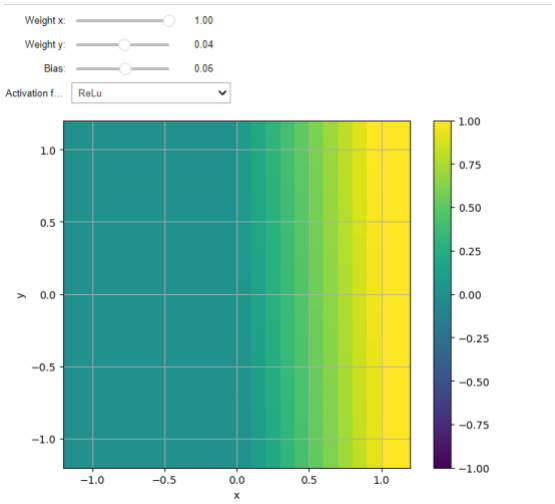
and -1.00. The yellow values change according tot he x-axis of the graph and gets closer to the origin (arround x-axis value of 1).



The same will happen with the y weight. Fort he bias, he shifts the decision boundary away from the origin. If the bias is positive, the boundary shifts one way; if negative, it shifts the opposite way.

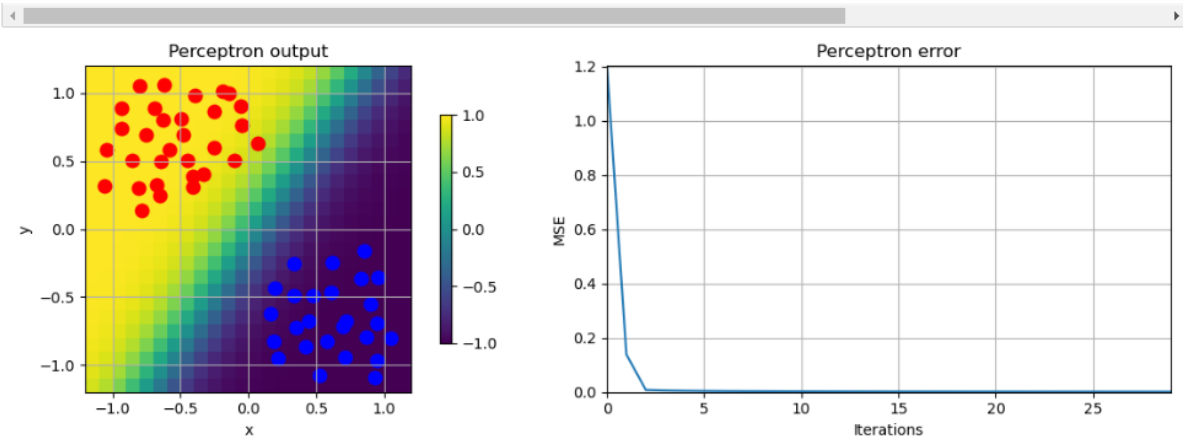
Here is a description for each activation function :

- **Linear**: The output is a plane in the input space, and modifying the weights will change the orientation
- **Sigmoid** : Outputs are squashed to between 0 and 1 and the is no more blue values (negative).
- **Hyperbolic Tangent** : Similar to the sigmoid but outputs between -1 and 1, resulting in a steeper decision boundary compared to the sigmoid.
- **ReLU** : It will clip all negative values to zero, resulting in a decision boundary that is linear and only affects one half of the input space. As seen below.



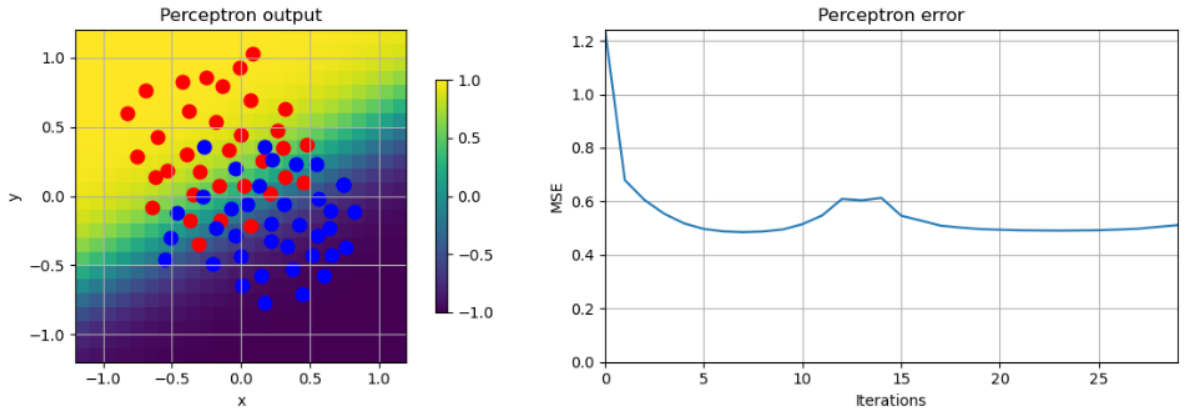
4_1_delta_rules_point

1. What happens if the boundaries between both classes are well defined



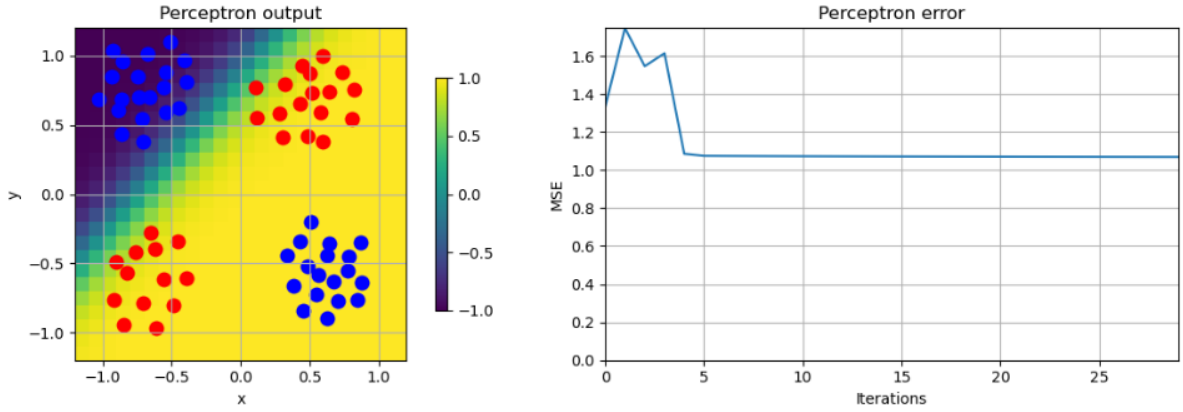
When class boundaries are clearly defined, they can be easily distinguished by the activation function's boundary. In such scenarios, as illustrated in the left image, when a data point falls on either side of the boundary, the model readily classifies it into the corresponding class. Consequently, the Mean Squared Error (MSE) tends to reach zero rapidly, reflecting the clear separation between the classes.

2. What happens if the classes overlap ? What could you say about oscillations in the error signal ?



In cases where class boundaries overlap, the classification becomes more challenging has we can see in the left image. This overlap leads to fluctuations in the error signal (right image). These oscillations occur because the model struggles to find a consistent boundary that separates the classes that we have choosen, causing variability in error as it attempts to adjust and adapt to the data. Still the MSE is not that high and the classification can be done well.

3. What happens if it is not possible to separate the classes with a single line ? What could you say about local minima ?



When it's impossible to separate classes with a single line, it indicates a non-linear relationship between the classes. In such cases, the model encounter local minima[1] during the training process. As a result, the model becomes "trapped" in these local minima, prevent its ability to find the most effective solution for separating the classes. The displayed images illustrate this challenge well. They show the function's attempt to seperate has possible the classes using a basic approach. However, despite these efforts, the MSE remains significantly high, indicative of the model being ensnared in a local minimum.

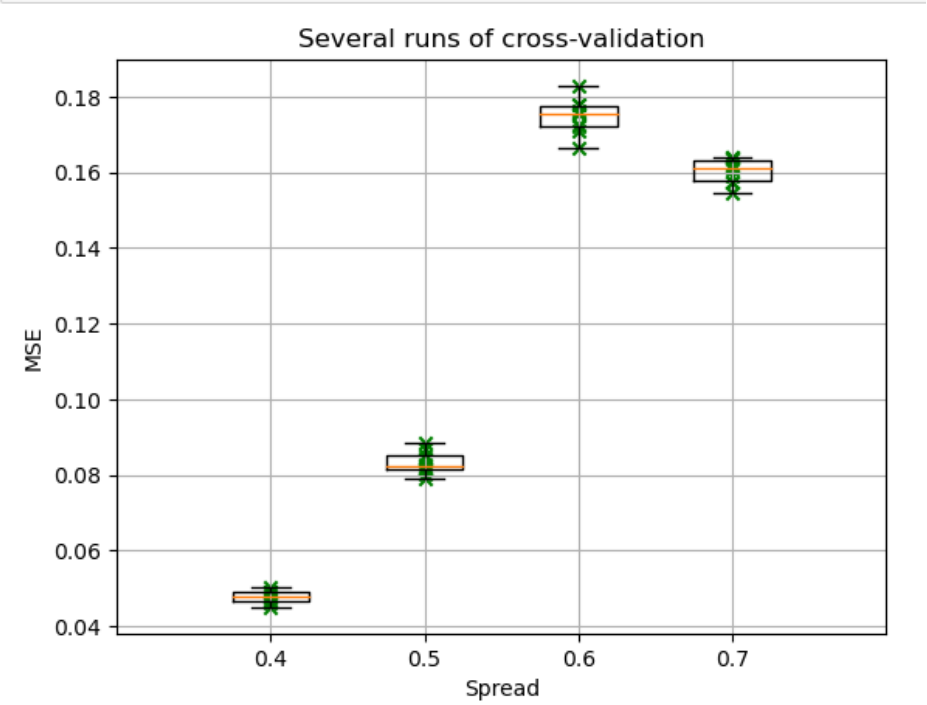
Backpropagation

Nothing to do – only launch the notebook plus understand the code and play with the parameters.

[1] A local minimum is a point where the model's parameters lead to a low error, but not the lowest possible error (global minimum)

Cross Validation

8_cross_validation



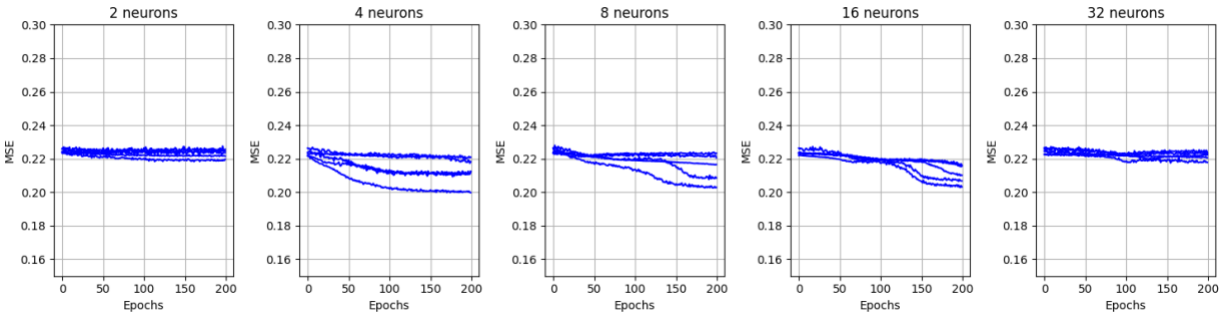
It our case, at lower spread values, the classes are well-separated, making the classification task easier for the model. This is reflected in the lower MSE values, indicating higher model accuracy.

As the spread increases, the classes begin to overlap more, which makes classification more difficult. This is observable in the graph as the median MSE increases and the range of MSE values becomes larger, showing more variability in the model's performance (not mush but still).

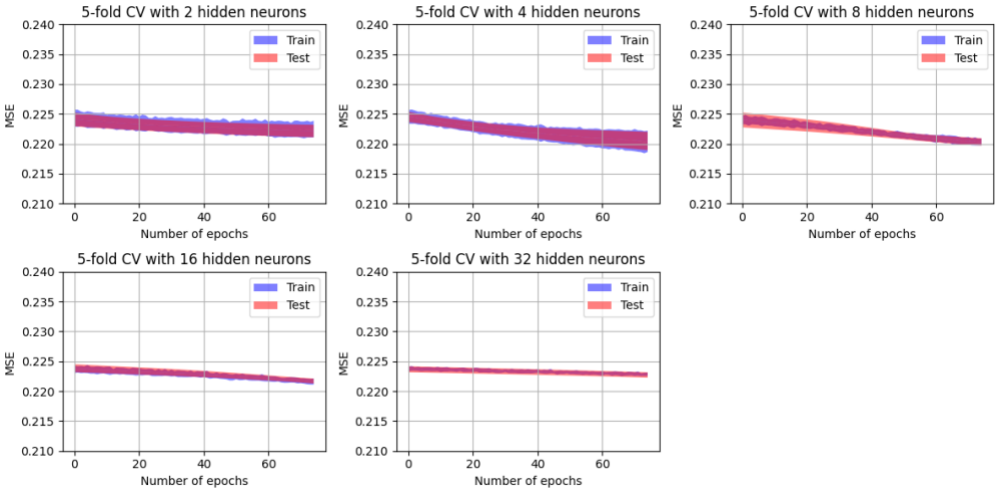
At the highest spread values, the overlap between classes presents a challenging classification scenario, which is evident from the higher and more spread out MSE values that we can observe in the image.

Model Building

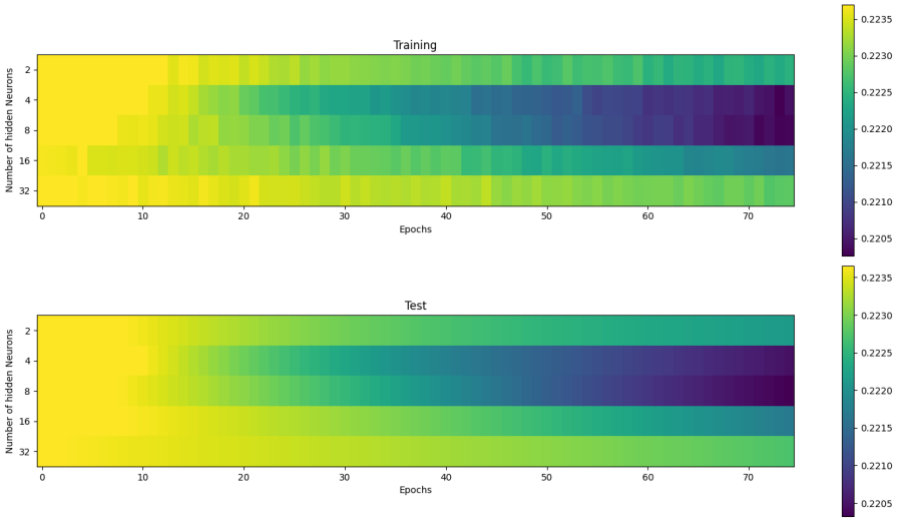
9_model_selection



The point at which the MSE stabilizes and stops decreasing significantly is around **100**.



The plot indicates that as the number of hidden neurons increases, the MSE decreases and stabilizes. However, more neurons also mean a more complex model, which can lead to overfitting. We'd want to choose a number that is low enough to prevent overfitting but high enough to capture the complexity of the data. It seems that the variance does not drastically reduce after 4 neurons.



The colorbar shows that the hidden neuron with less difference between the training and test is **4** which joins what says below.

Final Model

```
n_classes = len(label_binarizer.classes_) # 3: 'r', 'n', 'w'

n_features = dataset.shape[1] - n_classes # Number of features (excluding one-hot encoded labels)

EPOCHS = 100

K = 5
```

```
nn = mlp.MLP_N_output_classes([n_features,16,n_classes], 'softmax')

MSE_train, MSE_test, conf_mat = cv.k_fold_cross_validation(nn, dataset, k=K, learning_rate=LEARNING_RATE, momentum=MOMENTUM, epochs=EPOCHS, threshold=0.0)
```

```
def compute_accuracy(CM):

correct_predictions = np.trace(CM) # Sum of diagonal elements

total_predictions = np.sum(CM) # Sum of all elements

accuracy = correct_predictions / total_predictions

return accuracy
```

```
print('MSE training: ', MSE_train)

print('MSE test: ', MSE_test)

print('Confusion matrix:')

print(conf_mat)

print('Accuracy:', compute_accuracy(conf_mat))
```

```
MSE training:  0.2199178723317993
MSE test:  0.21970538874815948
Confusion matrix:
[[  74.  227. 1044.]
 [   3.   49. 1293.]
 [   0.   15. 1330.]]
Accuracy: 0.3600991325898389
```

(Reminder) The final model of Random Forest gave **86%**.

Conclusion

36% seems far too low and counter-performing. It's more likely to be a model implementation error (we've done a lot of tests without achieving any significant improvement) than an MLP feature.

Note that we used the **softmax** activation function because we're dealing with a multi-class problem and undersampled our dataset because it is unbalanced with the ‘r’ class being way less used (all classes with the same amount: 1345).