**Neural Networks:**

**Basic Introduction:**

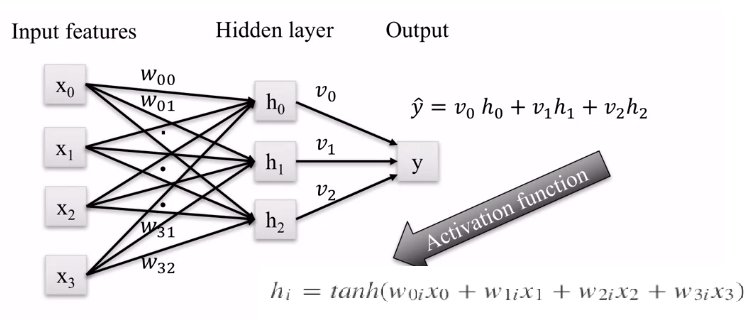
Neural networks are a type of machine learning in the field of deep learning, which is a subset of machine learning. Neural networks can be used to classify images, forecasting of continues values, and many more.

Neural networks and deep learning are such a complex and in-depth topic that it really requires its own course. The lecturer suggests takin this course:

<https://www.coursera.org/specializations/deep-learning?#courses>

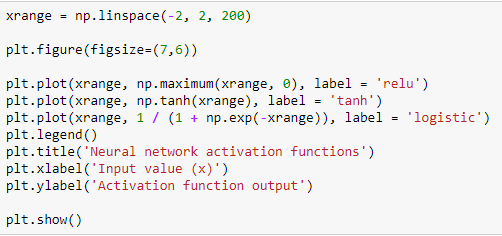
**Multi-layer Perceptron:**

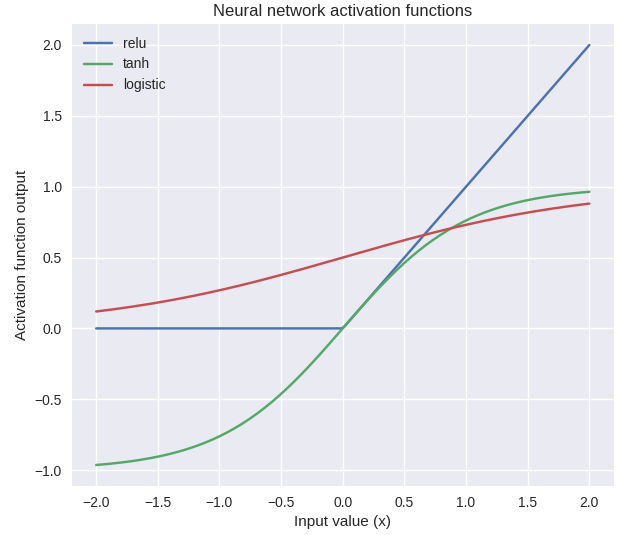
Let’s look at a type of neural network called a **Multi-layer Perceptron (MLP)**, these are also known as **feed forwards neural networks**. The example looks at a **single hidden layer**.



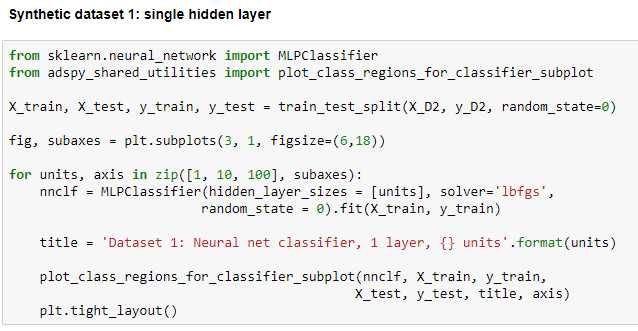
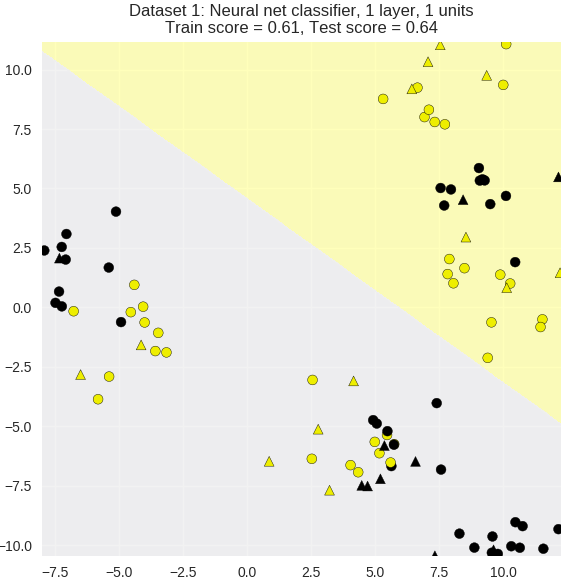
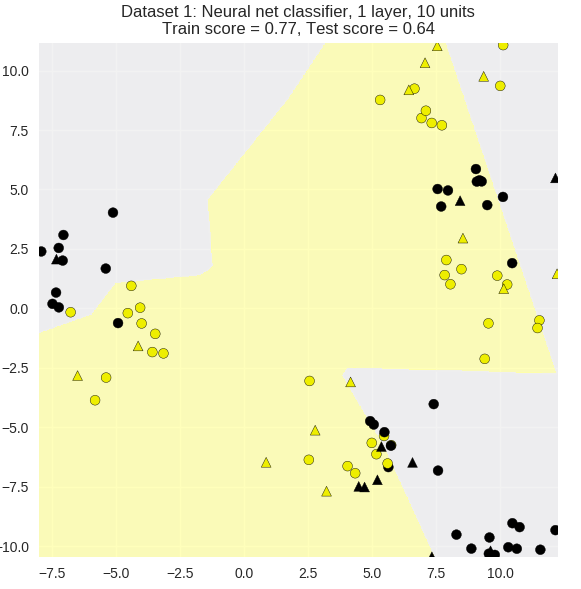
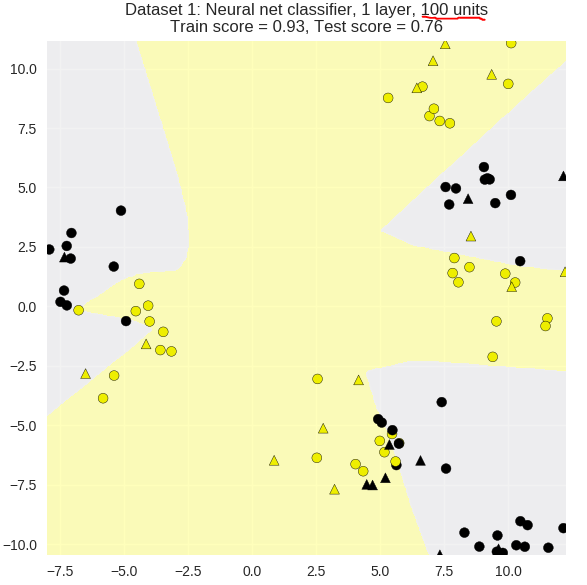
In the example above all the features are multiplied by weights these sum of weights and features are then put into an activation function which computes some value, this could be linear or non-linear. E.g. if we used the sigmoid function as our activation function then the possible values of h would be between 0-1. These values of h and weighted values are then summed and multiplied to give a number which can be used for classification or regression. The values of are found through the training process to ensure that the value y at the end classifies or predicts the desired training value as well as possible.

**Some Common Activation Functions Used:**





The default activation function in Scikit-learn is the **Relu function**, which is a function that return 0 if the input is <= 0 and then a simple y=x for any positive function. The **Sigmoid function** (logistic) returns a value between 0-1, and the **tanh function** returns a value of -1 to 1, with both of these centred around 0.



As the number of units increase (number of h’s) the model becomes more complex.

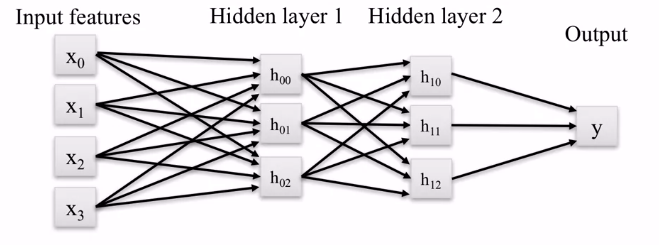
It’s possible to have an MLP with more **than 1 layer**, this is done by passing a **hidden layer sizer parameter** with multiple entries.

The parameter “**solver**” used above is used to learn the different **weights** of the network.

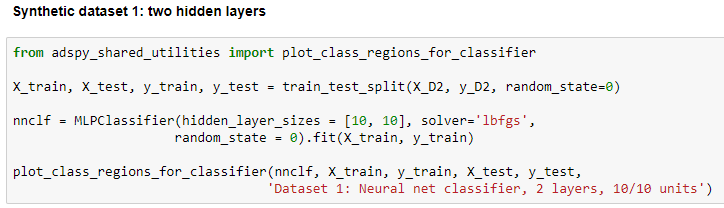
Also note that we’re passing in a **random state parameter**, this is because with a neural network the **initial weights are all random**, this can affect the learning process.

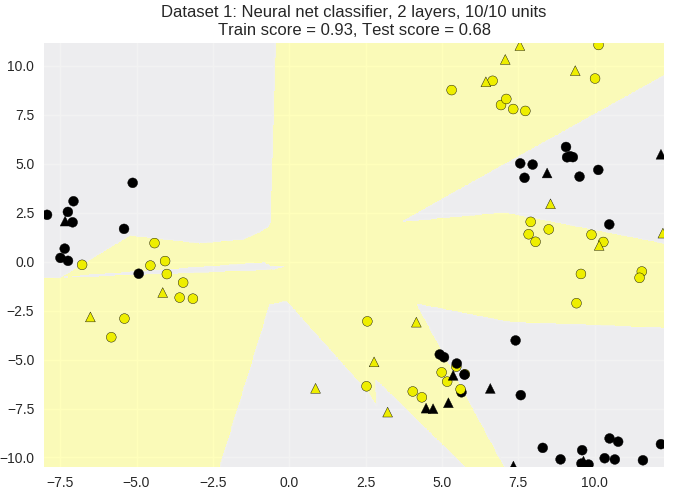
We can see from the above figures that by increasing the number of units in a layer we can increase the model’s complexity. For the example above we could actually still increase the models performance and complexity.

**Multi-layer Perceptron with Two Hidden Layers:**



The code below shows how to create a two-layer MLP:





Note: to used **two hidden layers** in our MLP we just **pass a list of value to the hidden later size parameter**.

Compare the above 2 hidden layer results to the plot with 1 hidden layer above (both for units=10), the 2-layer MLP has much better scores on the training and testing data. However, the computation required for adding additional layers is immense!

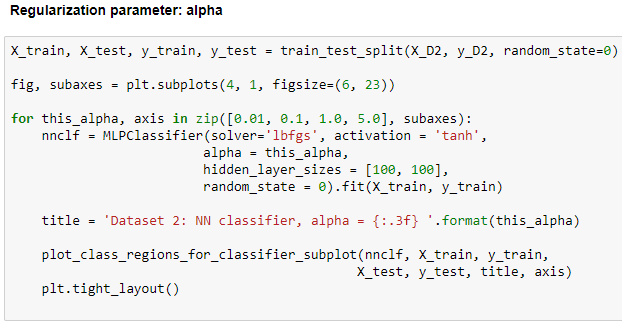
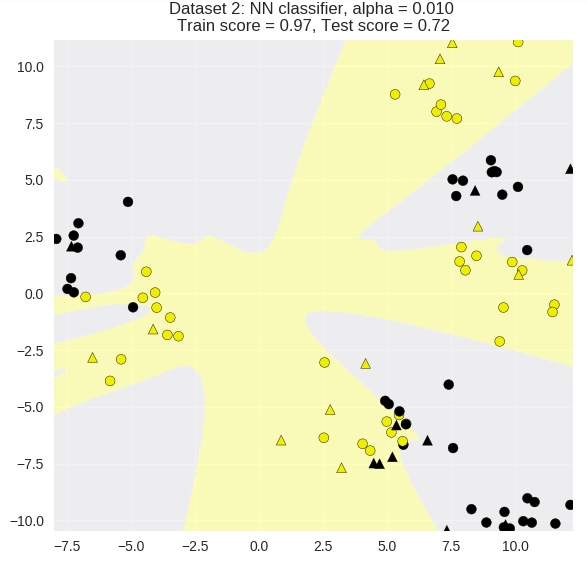
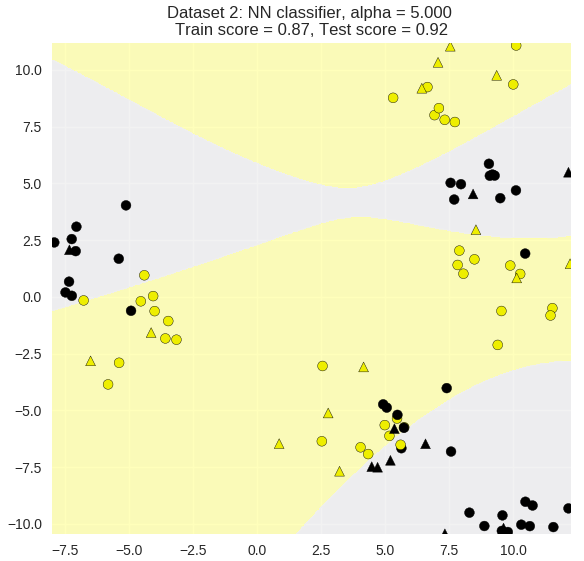
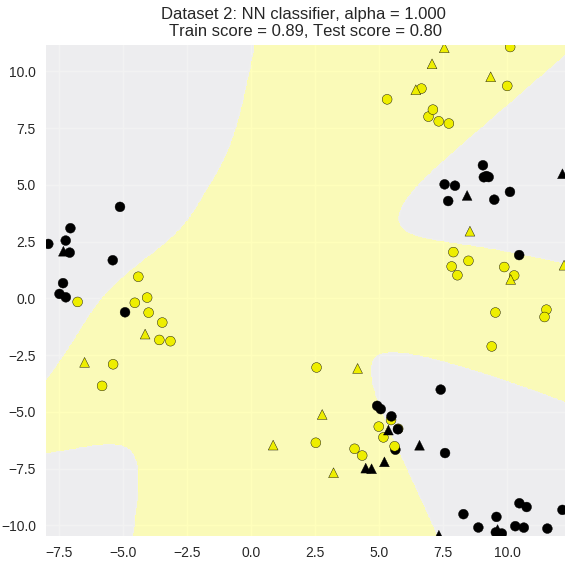
We can control the complexity of the model by applying an **L2 regularization penalty on the weights**. Remember that L2 regularization **penalizes models that have large sum of squares of all the weight values**. The result is that the process prefers models with **weights closer to zero**.

**L2 regularization (Alpha):**

This value is by default set to a small value 0.0001, that gives very little regularization.

The example below looks at the effects of a small to large alpha value on the synthetic data:

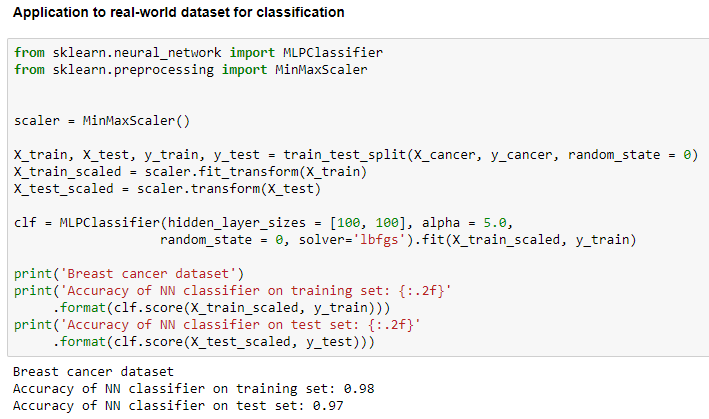
* Small alpha: more complex model, potential for overfitting.
* Large alpha: less complex model, potential for underfitting.



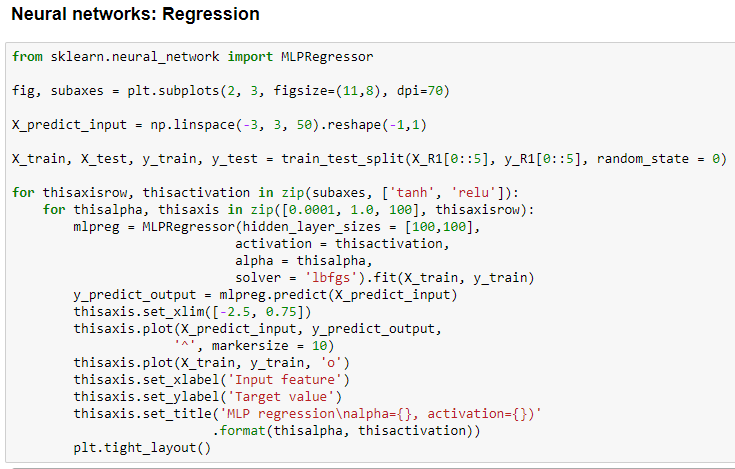
The left model shows the resulting decision boundaries from a small amount of regularization resulting in a more complex model with high training scores. The right shows a model with large alpha, so a less complex model, with a good balance of training and testing scores.

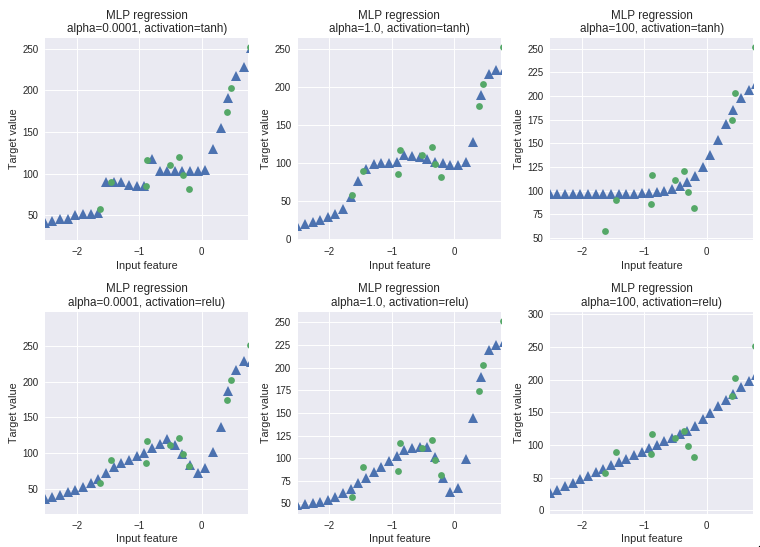
**It is critical to properly normalize the input features for Neural Networks!**

**Example using Scaled Real Data:**

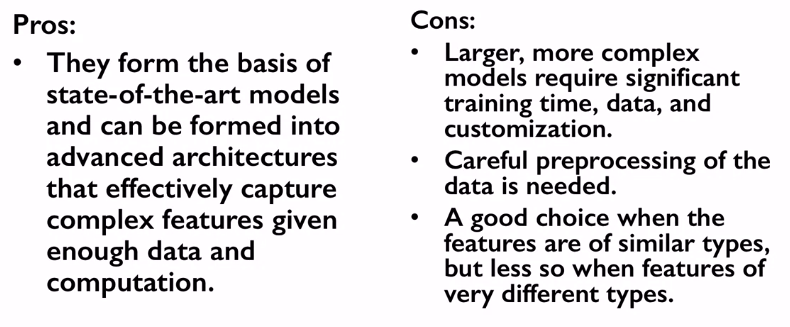


**Regression using MLP:**





Above the triangles are the predicted value and the circles are the training values (true values). We can see that by increase in the alpha regularization (left to right) we can see that with high regularization our model becomes simpler and smoother.



**How to control model complexity:**

